

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS





IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problem Mailbox.**

METHOD FOR RECOGNIZING CHARACTER AND DEVICE THEREFOR

Patent number: JP6068301
Publication date: 1994-03-11
Inventor: WANG SHIN-YWAN; VAEZI MEHRZAD R; SHERRICK CHRISTOPHER A
Applicant: CANON INC.; CANON INF SYST INC
Classification:
 - international: G06K9/20; G06K9/32; G06K9/34
 - european:
Application number: JP19930121883 19930426
Priority number(s):

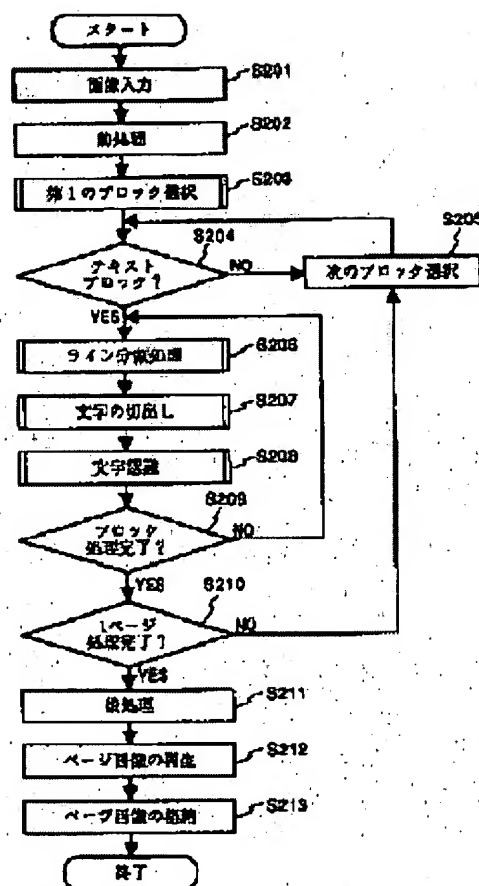
Also published as:

 EP0567344 (A2)
 US5680479 (A1)
 EP0567344 (A3)
 EP0567344 (B1)

Abstract of JP6068301

PURPOSE: To provide a character recognition method and device for recognizing a character on a document with a high speed and exactness, and preparing a text file.

CONSTITUTION: Pixel picture data of pixel units are inputted to a device (S201), the picture quality of the picture data is improved by a filter in order to improve deteriorated characters or pictures (S202), the hierarchical tree structure of the pictures is prepared (S203) so that text information/graphics information/ line information/picture information of information in a block can be identified, and each part in the picture can be reproduced in a proper sequence like a post-processing (S212), and whether or not the selected block is a text block is checked (S204). Then, a line dividing processing is operated to the text block (S206), each character in the line is segmented from another character in the line (S207), whether or not the processing to the text block and the processing to one page is ended is checked (S209 and 210), and a correction processing from an overall point of view such as context check or spelling check is operated (S211).



Data supplied from the esp@cenet database - Worldwide

T S1/5/1

1/5/1

DIALOG(R)File 347:JAPIO

(c) 2004 JPO & JAPIO. All rts. reserv.

04424401

METHOD AND DEVICE FOR RECOGNIZING CHARACTER

PUB. NO.: 06-068301 [JP 6068301 A]

PUBLISHED: March 11, 1994 (19940311)

INVENTOR(s): SHIN YAN WANGU

MEZAADOU AARU BAEZUII

KURISUTOFUAA EE SHIERITSUKU

APPLICANT(s): CANON INC [000100] (A Japanese Company or Corporation), JP
(Japan)CANON INF SYST INC [000000] (A Non-Japanese Company or
Corporation), US (United States of America)

APPL. NO.: 05-121883 [JP 93121883]

FILED: April 26, 1993 (19930426)

PRIORITY: 7-873,012 [US 873012-1992], US (United States of America),
April 24, 1992 (19920424)

INTL CLASS: [5] G06K-009/20; G06K-009/32; G06K-009/34

JAPIO CLASS: 45.3 (INFORMATION PROCESSING -- Input Output Units); 44.7
(COMMUNICATION -- Facsimile)JAPIO KEYWORD: R002 (LASERS); R107 (INFORMATION PROCESSING -- OCR & OMR
Optical Readers); R108 (INFORMATION PROCESSING -- Speech
Recognition & Synthesis); R131 (INFORMATION PROCESSING --
Microcomputers & Microprocessors); R139 (INFORMATION
PROCESSING -- Word Processors)

?

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平6-68301

(43) 公開日 平成6年(1994)3月11日

(51) Int.Cl. ⁵	識別記号	片内整理番号	F I	技術表示箇所
G 0 6 K 9/20	3 4 0 L			
9/32				
9/34				

審査請求 未請求 請求項の数286(全 420 頁)

(21) 出願番号 特願平5-121883

(22) 出願日 平成5年(1993)4月26日

(31) 優先権主張番号 8 7 3 0 1 2

(32) 優先日 1992年4月24日

(33) 優先権主張国 米国 (US)

(71) 出願人 000001007

キヤノン株式会社

東京都大田区下丸子3丁目30番2号

(71) 出願人 592208172

キヤノン インフォメーション システム
ズ インク.

Canon Information S
ystems, Inc.

アメリカ合衆国 カリフォルニア州

92626, コスタ メサ, ブルマン スト

リート 3188

(74) 代理人 弁理士 大塚 康徳 (外1名)

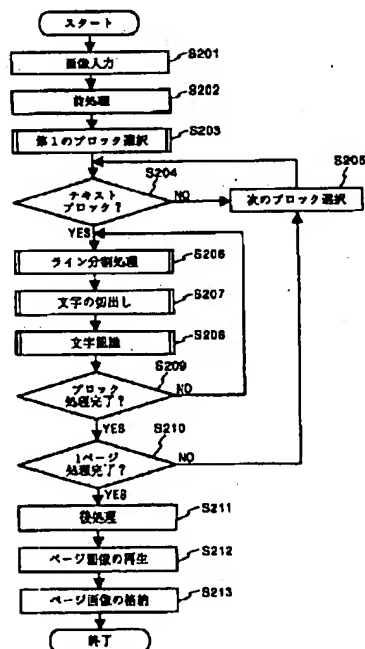
最終頁に続く

(54) 【発明の名称】 文字認識方法及び装置

(57) 【要約】 (修正有)

【目的】 高速かつ正確に文書上の文字を認識してテキストファイルを作成する文字認識方法及び装置の提供。

【構成】 画素単位の画素画像データを装置に入力し (S201)、フィルタで画像データの画質を向上して劣化した文字や画像を改善し (S202)、ブロック中情報のテキスト情報/グラフィックス情報/線画情報/画像情報などの識別と画像中の各部分を後処理 (S212) のような適切な順序で再生できるように画像の階層的な構造を生成し (S203)、選択されたブロックがテキストブロックか否かを調べ (S204)、テキストブロックに対してライン分割処理を行い (S206)、ライン中の各文字をライン中の他の文字から切り出し (S207)、テキストブロックに対する処理と1ページ分に対する処理が終了したかを調べ (S209, 210)、文脈チェックやスペルチェックなどの全体的な見地からの修正処理を行う (S211)。



1

【特許請求の範囲】

【請求項1】 画素画像データから画素ブロックを選別する方法であって、

画素データ内で連結要素の輪郭を追跡する行程と、
輪郭が追跡された各連結要素を囲んで矩形を形成する行程と、

サイズと他の矩形への近接関係とに基づいて矩形を選択的に横方向に連結して、ラインを形成する第1の連結行程と、

サイズと他のラインへの近接関係とに基づいてラインを選択的に縦方向に連結して、ブロックを形成する第2の連結行程とを備える方法。

【請求項2】 更に、画素画像データを入力する行程を備え、

該入力行程は、画素画像データが2値画素画像データでない場合に、画素画像データを2値画素画像データに変換する行程を含む請求項1記載の方法。

【請求項3】 前記矩形形成行程では、各連結要素を囲む最小の矩形が形成される請求項1記載の方法。

【請求項4】 更に、前記矩形形成行程で形成された対応する矩形の位置に基づいて、前記輪郭追跡行程で追跡された連結要素の階層の木構造を形成する行程を備える請求項1記載の方法。

【請求項5】 更に、前記矩形形成行程で形成された矩形をテキスト要素と非テキスト要素とに分類する行程を備える請求項4記載の方法。

【請求項6】 更に、テキスト要素のブロック内の文字画像を認識する行程を備える請求項5記載の方法。

【請求項7】 前記分類行程は、要素の高さの所定のしきい値に対応して実行される請求項5記載の方法。

【請求項8】 前記分類行程は、前記矩形形成行程で形成された矩形の高さに関連する統計情報に対応して実行される請求項6記載の方法。

【請求項9】 前記第1の連結行程と第2の連結行程とは、非テキスト要素に対しては実行されない請求項6記載の方法。

【請求項10】 更に、非テキスト要素に対し白輪郭を得る行程を備える請求項6記載の方法。

【請求項11】 非テキスト要素は白輪郭の数に対応して表要素と見なされる請求項10記載の方法。

【請求項12】 更に、白輪郭の充填率を算出する行程を備える請求項10記載の方法。

【請求項13】 非テキスト要素は、充填率が高い場合には画像データと見なされない請求項12記載の方法。

【請求項14】 更に、非格子状に配置された白輪郭を再接続する行程を備える請求項12記載の方法。

【請求項15】 非テキスト要素は、再接続率が高くない場合に表と見なされる請求項14記載の方法。

【請求項16】 白輪郭は4方向で算出される請求項10記載の方法。

2

【請求項17】 連結要素の輪郭は、少なくとも8方向に追跡される請求項1記載の方法。

【請求項18】 前記輪郭追跡行程では、連結要素の外部のみで連結要素の輪郭が追跡される請求項1記載の方法。

【請求項19】 更に、画素画像データ内でギャップを検出する行程を備え、前記第1の連結行程では、ギャップが矩形を隔てている場合には矩形がラインに連結されない請求項1記載の方法。

10 【請求項20】 矩形の間を縦に延びるギャップに対応して段を検出する請求項19記載の方法。

【請求項21】 前記第2の連結行程は、前記第1の連結行程で連結されたテキストデータのライン間に非テキストの境界を決定する行程を含み、前記第2の連結行程は、間に非テキストの境界がある場合にラインを縦方向にブロックとして連結しない請求項1記載の方法。

【請求項22】 更に、前記輪郭追跡行程の前に画素画像データを圧縮する行程を備える請求項1記載の方法。

【請求項23】 画素画像データ内の文字を認識する方法であって、

画素画像データ内で連結要素の輪郭を追跡する行程と、
追跡された連結要素がテキスト要素を含むか非テキスト要素を含むかを定める行程と、
テキスト要素を横方向に選択的に連結してテキストラインを形成する行程と、
テキストラインを選択的に縦方向に連結してテキストブロックを形成する行程とを含み、画素画像データから画素ブロックを選別する行程と、

テキストブロックを画素画像データのラインに分割する行程と、

30 前記分割行程で分割されたラインから文字を切り出す行程と、

前記切り出行程で切り出された文字を認識する行程と、

前記認識行程で認識された文字を、前記ブロック選別行程で確定された順に対応して格納する行程とを備える方法。

【請求項24】 更に、テキスト要素のブロック内の文字画像を認識する行程を備える請求項23記載の方法。

【請求項25】 更に、画素画像データが前処理される前処理行程を備える請求項23記載の方法。

40 【請求項26】 前記前処理行程は画像圧縮行程を含む請求項25記載の方法。

【請求項27】 前記前処理行程は画素画像データの画質を向上するフィルタ行程を含む請求項25記載の方法。

【請求項28】 更に、前記認識行程で認識された文字を後処理する行程を備える請求項23記載の方法。

【請求項29】 前記後処理は文脈チェックを含む請求項28記載の方法。

50 【請求項30】 更に、画素画像データを入力する行程を備え、

3

該入力行程は、画素画像データが2値画素画像データでない場合に、画素画像データを2値画素画像データに変換する行程を含む請求項2記載の方法。

【請求項31】 前記ブロック選別行程は、前記輪郭追跡行程で定められた連結画素に基づいて階層的木構造を形成する行程を含む請求項2記載の方法。

【請求項32】 更に、前記輪郭追跡行程で追跡された連結要素を囲む矩形を形成する行程を備える請求項31記載の方法。

【請求項33】 前記階層的木構造は、前記矩形形成行程で形成された矩形の位置に基づいて形成される請求項32記載の方法。

【請求項34】 更に、第1及び第2の連結行程に基づいて、階層的木構造を更新する行程を備える請求項31記載の方法。

【請求項35】 更に、連結要素をテキスト要素又は非テキスト要素に分類する行程を備える請求項31記載の方法。

【請求項36】 前記第1及び第2の連結行程は、テキスト要素を連結するが非テキスト要素は連結しない請求項35記載の方法。

【請求項37】 更に、非テキスト要素の内部で白輪郭を追跡する行程を備える請求項35記載の方法。

【請求項38】 更に、非テキスト要素に含まれる白輪郭の数に基づいて、非テキスト要素を表と見なす行程を備える請求項37記載の方法。

【請求項39】 更に、非格子状に配置された白輪郭を再接続する行程を備える請求項37記載の方法。

【請求項40】 前記非テキスト要素は、前記再接続行程での再接続に基づいて表と見なされる請求項39記載の方法。

【請求項41】 更に、非テキスト要素内の白輪郭の充填率を算出する行程を備える請求項35記載の方法。

【請求項42】 非テキスト要素は、充填率が高い場合に表と見なされない請求項41記載の方法。

【請求項43】 画素画像データ内で文字のテキストファイルを形成する方法であって、

画素画像データが2値の画素画像データでない場合に、画素画像データを2値の画素画像データに変換する行程を含む画素画像データの入力行程と、

画素画像データ内で連結要素の輪郭を追跡する行程と、追跡された連結要素のサイズに基づいて連結要素がテキスト要素か非テキスト要素かを定める行程と、隣接するテキスト要素の近接関係に基づいてテキスト要素を選択的に横方向に連結してテキストラインを形成する行程と、隣接するテキストラインの近接関係とテキスト要素間の非テキスト要素の位置とに基づいてテキストラインを選択的に縦方向に連結してテキストブロックを形成する行程とを含み、画素画像データのブロックを選別する行程と、

4

前記輪郭追跡で追跡された画素要素に基づいて、階層的木構造を形成する行程と、

テキストブロックを画素画像データのラインに分割する行程と、

前記分割行程で分割されたラインから文字を切り出す行程と、

前記切り出行程で切り出された文字を認識し、前記認識に基づいて文字コードを割り当てる行程と、

前記階層的木構造で確定された順に対応して文字コードをコンピュータのテキストファイルに格納する行程とを備える方法。

【請求項44】 更に、画素画像データが前処理される前処理行程を備える請求項43記載の方法。

【請求項45】 前記前処理行程は画像圧縮行程を含む請求項44記載の方法。

【請求項46】 前記前処理行程は画素画像データの画質を向上する行程を含む請求項45記載の方法。

【請求項47】 更に、文脈チェックを含み、前記認識行程で認識された文字を後処理する行程を備える請求項43記載の方法。

【請求項48】 更に、非テキスト要素の内部で白輪郭を追跡する行程を備える請求項43記載の方法。

【請求項49】 更に、非テキストに識別子を付与する行程を更に備える請求項48記載の方法。

【請求項50】 更に、非テキスト要素に含まれる白輪郭の数に基づいて表識別子を付与する行程を備える請求項49記載の方法。

【請求項51】 更に、非格子状配置の白輪郭を再接続する行程を備え、非テキスト要素には前記再接続行程での再接続率に基づいて表の識別子が付与される請求項49記載の方法。

【請求項52】 更に、非テキスト要素内の白輪郭の充填率を算出する行程を備え、充填率が低い場合に非テキスト要素に表の識別子を付与する請求項49記載の方法。

【請求項53】 画素画像データから画素ブロックを選別する装置であって、

画素データ内で連結要素の輪郭を追跡する輪郭追跡手段と、

前記輪郭追跡手段によって追跡された各連結要素を囲んで矩形を形成する矩形形成手段と、

サイズと他の矩形への近接関係とに基づいて矩形を選択的に横方向に連結して、ラインを形成する第1連結手段と、

サイズと他のラインへの近接関係に基づいてラインを選択的に縦方向に連結して、ブロックを形成する第2連結手段とを備える装置。

【請求項54】 更に、画素画像データを入力する入力手段を備え、

50 該入力手段は、画素画像データが2値画素画像データで

5

ない場合に、画素画像データを2値画素画像データに変換する変換手段を含む請求項53記載の装置。

【請求項55】 前記矩形形成手段では、各連結要素を囲む最小の矩形が形成される請求項53記載の装置。

【請求項56】 更に、前記矩形形成手段で形成された対応する矩形の位置に基づいて、前記輪郭追跡手段で追跡された連結要素の階層の木構造を形成する第2の形成手段を備える請求項53記載の装置。

【請求項57】 更に、前記矩形形成手段で形成された矩形をテキスト要素と非テキスト要素とに分類する分類手段を備える請求項56記載の装置。

【請求項58】 更に、テキスト要素のブロック内の文字画像を認識する認識手段を備える請求項57記載の装置。

【請求項59】 前記分類手段は、要素の高さの所定のしきい値に対応して分類する請求項57記載の装置。

【請求項60】 前記分類手段は、前記矩形形成手段で形成された矩形の高さに関連する統計情報に対応して分類する請求項57記載の装置。

【請求項61】 前記第1連結手段と第2連結手段とは、非テキスト要素には使用されない請求項57記載の装置。

【請求項62】 更に、非テキスト要素に対して白輪郭を得る手段を備える請求項57記載の装置。

【請求項63】 非テキスト要素は白輪郭の数に対応して表の要素と見なされる請求項62記載の装置。

【請求項64】 更に、白輪郭の充填率を算出する算出手段を備える請求項62記載の装置。

【請求項65】 非テキスト要素は、充填率が高い場合には画像データと見なされない請求項64記載の装置。

【請求項66】 更に、非格子状に配置された白輪郭を再接続する再接続手段を備える請求項64記載の装置。

【請求項67】 非テキスト要素は、再接続率が低い場合に表と見なされる請求項66記載の装置。

【請求項68】 白輪郭は4方向で算出される請求項62記載の装置。

【請求項69】 連結要素の輪郭は、少なくとも8方向に追跡される請求項53記載の装置。

【請求項70】 前記輪郭追跡手段では、連結要素の外部のみで連結要素の輪郭が追跡される請求項53記載の装置。

【請求項71】 更に、画素画像データ内でギャップを検出する検出手段を備え、前記第1連結手段では、ギャップが矩形を隔てている場合には矩形がラインに連結されない請求項53記載の装置。

【請求項72】 矩形の間を縦に延びるギャップに対応して段が検出される請求項71記載の装置。

【請求項73】 前記第2連結手段は、前記第1連結手段で連結されたテキストデータのライン間に非テキストの境界を決定する決定手段を含み、前記第2連結手段

6

は、間に非テキストの境界がある場合にラインを縦方向にブロックとして連結しない請求項53記載の装置。

【請求項74】 更に、前記輪郭追跡手段の前に、画素画像データを圧縮する圧縮手段を備える請求項53記載の装置。

【請求項75】 画素画像データ内の文字を認識する装置であって、

画素画像データ内で連結要素の輪郭を追跡する行程と、定められた連結要素がテキスト要素を含むか非テキスト要素を含むかを定める行程と、テキスト要素を横方向に選択的に連結してテキストラインを形成する行程と、テキストラインを選択的に縦方向に連結してテキストブロックを形成する行程とを含み、画素画像データから画素ブロックを選別するブロック選別手段と、テキストブロックを画素画像データのラインに分割する分割手段と、

前記分割手段で分割されたラインから文字を切り出す切出手段と、

前記切出手段で切り出された文字を認識する認識手段と、

前記認識手段で認識された文字を、前記ブロック選別手段で確定された順に対応して格納する格納手段とを備える装置。

【請求項76】 更に、テキスト要素のブロック内の文字画像を認識する第2の認識手段を備える請求項75記載の装置。

【請求項77】 更に、画素画像データが前処理される前処理手段を備える請求項75記載の装置。

【請求項78】 前記前処理手段は画像圧縮手段を含む請求項77記載の装置。

【請求項79】 前記前処理手段は画素画像データの画質を向上するフィルタ手段を含む請求項77記載の装置。

【請求項80】 更に、前記認識手段で認識された文字を後処理する後処理手段を備える請求項75記載の装置。

【請求項81】 前記後処理手段は文脈チェック手段を含む請求項80記載の装置。

【請求項82】 更に、画素画像データを入力する入力手段を備え、

該入力手段は、画素画像データが2値画素画像データでない場合に、画素画像データを2値画素画像データに変換する変換手段を含む請求項75記載の装置。

【請求項83】 前記ブロック選別手段は、前記輪郭追跡手段で追跡された連結要素に基づいて階層の木構造を形成する手段を含む請求項75記載の装置。

【請求項84】 更に、前記輪郭追跡手段で定められた連結要素を囲む矩形を形成する形成手段を備える請求項83記載の装置。

【請求項85】 前記階層の木構造は、前記矩形形成手

段で形成された矩形の位置に基づいて形成される請求項84記載の装置。

【請求項86】 更に、第1及び第2連結手段に基づいて、階層の木構造を更新する更新手段を備える請求項64記載の装置。

【請求項87】 更に、連結要素をテキスト要素又は非テキスト要素に分類する分類手段を備える請求項83記載の装置。

【請求項88】 前記第1及び第2連結手段は、テキスト要素を連結するが非テキスト要素は連結しない請求項87記載の装置。

【請求項89】 更に、非テキスト要素の内部で白輪郭を追跡する手段を備える請求項87記載の装置。

【請求項90】 更に、非テキスト要素に含まれる白輪郭の数に基づいて表と見なす手段を備える請求項89記載の装置。

【請求項91】 更に、非格子状に配置された白輪郭を再接続する再接続手段を備える請求項89記載の装置。

【請求項92】 前記非テキスト要素は、前記再接続手段での再接続に基づいて表と見なされる請求項91記載の装置。

【請求項93】 更に、非テキスト要素内の白輪郭の充填率を算出する算出手段を備える請求項87記載の装置。

【請求項94】 非テキスト要素は、充填率が高い場合に表と見なされない請求項93記載の装置。

【請求項95】 画素画像データ内で文字のテキストファイルを形成する装置であって、

画素画像データが2値の画素画像データでない場合に、画素画像データを2値の画素画像データに変換する変換手段を含む画素画像データの入力手段と、

画素画像データ内で連結要素の輪郭を追跡する輪郭追跡手段と、取り出された連結要素のサイズに基づいて連結要素がテキスト要素か非テキスト要素かを定める決定手段と、隣接するテキスト要素の近接関係に基づいてテキスト要素を選択的に横方向に連結してテキストラインを形成する第1連結手段と、隣接するテキストラインの近接関係とテキスト要素間の非テキスト要素の位置とに基づいてテキストラインを選択的に縦方向に連結してテキストブロックを形成する第2連結手段とを含み、画素画像データのブロックを選別するブロック選別手段と、

前記輪郭追跡で追跡された画素要素に基づいて、階層の木構造を形成する形成手段と、

テキストブロックを画素画像データのラインに分割する分割手段と、

前記分割手段で分割されたラインから文字を切り出す切り出す手段と、

前記切り出す手段で切り出された文字を認識し、前記認識に基づいて文字コードを割り当てる認識手段と、

前記階層の木構造で確定された順に対応して文字コード

50

をコンピュータのテキストファイルに格納する格納手段とを備える装置。

【請求項96】 更に、画素画像データが前処理される前処理手段を備える請求項95記載の装置。

【請求項97】 前記前処理手段は画像圧縮手段を含む請求項96記載の装置。

【請求項98】 前記前処理手段は画素画像データの画質向上手段を含む請求項97記載の装置。

【請求項99】 更に、文脈チェック手段を含み、前記認識手段で認識された文字を後処理する後処理手段を備える請求項95記載の装置。

【請求項100】 更に、非テキスト要素の内部で白輪郭を追跡する手段を備える請求項95記載の装置。

【請求項101】 更に、非テキストに識別子を付与する付与手段を更に備える請求項100記載の装置。

【請求項102】 更に、非テキスト要素に含まれる白輪郭の数に基づいて、非テキスト要素に表の識別子が付与される請求項101記載の装置。

【請求項103】 更に、非格子状配置の白輪郭を再接続する再接続手段を備え、非テキスト要素には前記再接続手段での再接続率に基づいて表の識別子が付与される請求項101記載の装置。

【請求項104】 更に、非テキスト要素内の白輪郭の充填率を算出する算出手段を備え、充填率が高くない場合に非テキスト要素に表識別子を付与する請求項101記載の装置。

【請求項105】 画素画像データでラインの位置を割り出す方法であって、

画像データの横方向に伸びた部分の全域で画像密度の水平投影を得る行程と、

前記水平投影に基づいて、画像密度が投影された部分の幅を減少するか否かを決定する行程と、

前記水平投影に基づいて、ラインの位置を割り出す行程を備える方法。

【請求項106】 前記決定行程は、実質的に空白の部分で水平投影を調べる行程と、実質的に空白の部分が見付からない場合に幅を減少する行程とを備える請求項105記載の方法。

【請求項107】 前記調べる行程は、間隔の小さい非空白の部分を選択する行程を含む請求項106記載の方法。

【請求項108】 前記水平投影を得る行程と決定する行程とは、順に繰り返して実行される請求項105記載の方法。

【請求項109】 前記割り出す行程では、水平投影の実質的に空白の領域間の画像データのライン位置を、ラインとして割り出す請求項105記載の方法。

【請求項110】 更に、画素画像データを前処理する前処理行程を備える請求項105記載の方法。

【請求項111】 段に分割可能な画素画像データ内で

ラインの位置を割り出す方法であって、画像データの少なくとも1つの横方向に伸びる段の全域で、画像密度の水平投影を得る行程と、前記行程で得られた水平投影に基づいて、画像密度が投影される段の数を増加するか否かを決定する行程と、前記水平投影に基づいてデジタル画像データ内でラインの位置を割り出す行程とを備える方法。

【請求項112】 前記決定行程は、実質的に空白の部分で水平投影を調べる行程と、実質的に空白の部分が見付からない場合に、段の数を増加する行程とを備える請求項111記載の方法。

【請求項113】 前記調べる行程は間隔の小さい非空白の部分とを連結する行程を含む請求項112記載の方法。

【請求項114】 前記水平投影を得る行程と決定する行程とは、順に繰り返し実行される請求項111記載の方法。

【請求項115】 前記割出行程では、水平投影の実質的に空白の領域間の画像データのライン位置を、ラインとして割り出す請求項111記載の方法。

【請求項116】 更に、画像データの隣接する段のラインが重なるか否かを決定する行程を備える請求項111記載の方法。

【請求項117】 前記ラインが重なるかを決定する行程は、スケール不変の決定から成る請求項116記載の方法。

【請求項118】 更に、画素画像データを前処理する前処理行程を備え、前記前処理行程は画像縮小行程を含む請求項111記載の方法。

【請求項119】 前記画像縮小は水平方向と垂直方向とで異なる請求項118記載の方法。

【請求項120】 画素画像データのラインセグメントから文字画像を切り出す方法であって、ラインセグメントから非接触で重なっていない文字を切り出す第1の切出行程と、ラインセグメントから接触文字を切り出す第2の切出行程とを備える方法。

【請求項121】 更に、高解像度の画像セグメントを低解像度に圧縮する画像データのイメージセグメントの前処理行程を備える請求項120記載の方法。

【請求項122】 垂直方向と水平方向とで異なる圧縮率が使われる請求項121記載の方法。

【請求項123】 更に、前記第1の切出行程で切り出された文字を認識処理する行程を備え、前記第2の切出行程は前記認識行程で認識されなかった文字に対してのみ実行される請求項120記載の方法。

【請求項124】 更に、非接触で重なっている文字データをラインセグメントから切り出す中間の切出行程を備える請求項123記載の方法。

【請求項125】 前記中間切出行程は、非接触で重な

っている文字を画素データの輪郭の追跡により切り出す行程を含む請求項124記載の方法。

【請求項126】 前記中間切出行程は、間違っって切り出された文字を再接続する行程を含む請求項124記載の方法。

【請求項127】 更に、前記認識行程で認識されなかった文字を再接続する行程を備える請求項123記載の方法。

【請求項128】 前記第1の切出行程は、前記ラインセグメントを非空白の画素データが見付かるまでとびとびに処理する行程を含む請求項120記載の方法。

【請求項129】 更に、前記とびとびの処理行程で非空白の画素が見付かった場合に、緻密に前後をサーチする行程を備える請求項128記載の方法。

【請求項130】 前記第2の切出行程は非垂直切り出しによって接触文字を切り出す行程を含む請求項120記載の方法。

【請求項131】 非垂直切り出しの角度は画素密度の垂直投影特性に対応して決められる請求項130記載の方法。

【請求項132】 更に、垂直投影特性に基づいて少なくとも1つの回転された投影特性を得る行程を備えることを特徴とする131記載の方法。

【請求項133】 更に、前記垂直投影特性の角度に近接する角度で複数の回転された投影特性を得る行程を備える請求項132記載の方法。

【請求項134】 非垂直切り出しの角度は、前記回転投影特性を得る行程と垂直投影特性を得る行程とから得られた投影特性の最小値に基づいて得られる請求項133記載の方法。

【請求項135】 前記第2の切出行程は、前記ラインセグメント内の文字の特徴が既知かどうかに基づいて選択可能である請求項120記載の方法。

【請求項136】 前記第2の切出行程は、前記文字セット内の文字データのスペースの統計に基づいて切り出す行程を含む請求項135記載の方法。

【請求項137】 更に、小さな文字のセットを再接続する行程を備える請求項136記載の方法。

【請求項138】 文字画素データのラインセグメント内で接触文字を切り出す方法であって、

ラインセグメント内で画素データの垂直投影特性を得る行程と、

垂直投影特性の最小値から隣接する最大値への方向から少なくとも1つの角度を算出する第1の算出行程と、

前記第1の算出行程で算出された角度に基づいて、少なくとも1つの回転された投影特性を算出する第2の算出行程と、

前記回転投影特性の最小値に対応する位置で前記回転投影特性の角度にラインセグメントを切る行程とを備える

方法。

【請求項139】 前記第1の算出行程は、前記垂直投影特性を複数のしきい値と比較して前記最小値と隣接する最大値との位置を決定する行程を含む請求項138記載の方法。

【請求項140】 更に、最小値と隣接する最大値とが見付からない場合に、前記しきい値を増加させる行程を備える請求項139記載の方法。

【請求項141】 最小値は、少なくとも一方の側が第2しきい値より大きい最大値によって囲まれ、第1しきい値より小さな垂直投影特性上の位置として割り出される請求項139記載の方法。

【請求項142】 更に、各々が前記第2の算出行程で算出された回転投影特性に近接する角度で算出される複数の回転投影特性を算出する行程を備える請求項138記載の方法。

【請求項143】 更に、複数の回転投影特性と垂直投影特性との全体の最小値を割り出す行程を備え、前記切断行程は最小値の位置に対応する位置で最小値が得られた角度に対応する角度に切る請求項142記載の方法。

【請求項144】 画素画像データ内で文字を認識する

方法であって、
画素画像データから画素ブロックを選別する行程と、
少なくとも1つの段全体の画素密度の水平投影に基づいてテキストブロックを少なくとも1つの段に適応的に分配することにより、テキストブロックを画素データのラインに分割する行程と、

非接触で重なっていない文字がラインセグメントから切り出される第1の切出行程と、接触文字を切り出す少なくとも1つの追加の切出行程とを含み、前記分割行程で分割されたラインセグメントから文字を切り出す行程と、

前記文字切出行程で切り出された文字を認識する行程と、
前記認識行程で認識された文字を前記ブロック選別行程で確定された順に対応して格納する行程とを備える方法。

【請求項145】 更に、画素画像データが2値の画素画像データでない場合に、画素画像データを2値の画素画像データに変換する行程を含む画素画像データを入力する行程を備える請求項144記載の方法。

【請求項146】 更に、画素画像データが前処理される前処理行程を備える請求項144記載の方法。

【請求項147】 前記前処理行程は、画像圧縮行程を含む請求項146記載の方法。

【請求項148】 前記前処理行程は、画素画像データの画質を向上するフィルタ行程を含む請求項146記載の方法。

【請求項149】 更に、前記認識行程で認識された文字を後処理する行程を備える請求項144記載の方法。

【請求項150】 前記後処理は文脈チェックを含む請

求項149記載の方法。

【請求項151】 前記第1の切出行程はラインセグメントをとびとびに処理する行程を含む請求項144記載の方法。

【請求項152】 前記追加の切出行程は、画素画像データの文字について特徴が既知か否かに対応して選択可能である請求項144記載の方法。

【請求項153】 前記追加の切出行程では既知の文字セットの統計に基づいて切り出しを行う請求項152記載の方法。

【請求項154】 前記追加の切出行程では画素密度の回転投影に対応して切り出しを行う請求項152記載の方法。

【請求項155】 画素密度の回転投影は画素密度の垂直投影に基づいて決められた角度で得られる請求項154記載の方法。

【請求項156】 更に、前記第1の切出行程と前記少なくとも1つの追加の切出行程の間に中間の切出行程を備える請求項154記載の方法。

【請求項157】 前記中間の切出行程はラインセグメント内の重なった画素画像の輪郭を追跡する行程を含む請求項156記載の方法。

【請求項158】 更に、文字が間違っって切り出された場合に、前記中間の切出行程で切り出された文字を再接続する行程を備える請求項157記載の方法。

【請求項159】 前記認識行程は、前記追加の切出行程の前に前記第1の切出行程で切り出された文字を認識し、前記追加の切出行程は前記認識行程で認識されない文字に対して行なわれる請求項158記載の方法。

【請求項160】 更に、文字が前記認識行程で認識されない場合に、前記少なくとも1つの追加の切出行程で切り出された文字を再接続する行程を備える請求項159記載の方法。

【請求項161】 画素画像データから文字のテキストファイルを形成する方法であって、

画素画像データが2値の画素画像データでない場合に、画素画像データを2値の画素画像データに変換する行程を含む画素画像データの入力行程と、

画素画像データのブロックがテキストブロックか非テキストブロックかに対応して画素画像データのブロックを選別する行程と、

テキスト及び非テキストブロックの階層的木構造を得る行程と、

少なくとも1つの段の全域の画素密度の水平投影に基づいて、テキストブロックを前記少なくとも1つの段で適応的に分離することにより、テキストブロックを画素画像データのラインに分割する行程と、

非接触で重なっていない文字をラインセグメントから切り出す第1の切出行程と、接触文字を切り出す少なくとも1つの追加の切出行程とを含み、前記分割行程で分割

されたラインから文字を切り出す行程とを備える方法。

【請求項162】 更に、画素画像データが前処理される前処理行程を備える請求項161記載の方法。

【請求項163】 前記前処理行程は画像圧縮行程を含む請求項162記載の方法。

【請求項164】 前記前処理行程は画素画像データの画質を向上する行程を含む請求項163記載の方法。

【請求項165】 更に、文脈チェックを含み、前記認識行程で認識された文字を後処理する行程を備える請求項161記載の方法。

【請求項166】 前記第1の切出行程はラインセグメントをとびとびに処理する行程を含む請求項161記載の方法。

【請求項167】 前記追加の切出行程は、画素画像データの文字について特徴が既知か否かに対応して選択可能である請求項166記載の方法。

【請求項168】 前記追加の切出行程では既知の文字セットの統計に基づいて切り出しを行う請求項167記載の方法。

【請求項169】 前記追加の切出行程では画素密度の回転投影に対応して切り出しを行う請求項167記載の方法。

【請求項170】 画素密度の回転投影は画素密度の垂直投影に基づいて決められた角度で得られる請求項169記載の方法。

【請求項171】 更に、前記第1の切出行程と前記少なくとも1つの追加の切出行程の間に中間の切出行程を備える請求項161記載の方法。

【請求項172】 前記中間の切出行程はラインセグメント内の重なった画素画像の輪郭を追跡する行程を含む請求項171記載の方法。

【請求項173】 更に、文字が間違っって切り出された場合に、前記中間の切出行程で切り出された文字を再接続する行程を備える請求項172記載の方法。

【請求項174】 前記認識行程は、前記追加の切出行程の前に前記第1の切出行程で切り出された文字を認識し、前記追加の切出行程は前記認識行程で認識されない文字に対して行なわれる請求項173記載の方法。

【請求項175】 更に、文字が前記認識行程で認識されない場合に、前記少なくとも1つの追加の切出行程で切り出された文字を再接続する行程を備える請求項174記載の方法。

【請求項176】 画素画像データから文字のテキストファイルを形成する方法であって、画素画像データが2値の画素画像データでない場合に、画素画像データを2値の画素画像データに変換する行程を含み、画素画像データを入力する行程と、

画素画像データ内で連結要素の輪郭を追跡する行程と、前記追跡行程の間に追跡された連結要素のサイズに基づいて追跡された連結要素がテキスト要素を含むか非テキ

スト要素を含むから決める行程と、隣接するテキスト要素の近接関係に基づいてテキスト要素を選択的に横方向に連結してテキストラインを形成する行程と、隣接するテキストラインの近接関係とテキストライン間の非テキスト要素の位置とに基づいてテキストラインを選択的に縦方向に連結する行程とを含み、画素画像データのブロックを選別する行程と、

前記輪郭追跡行程で追跡された連結要素に基づいて、階層的木構造を形成する行程と、

10 少なくとも1つの段の全域での画素密度の水平投影に基づいて、テキストブロックを前記少なくとも1つの段に適応的に分離することによって、テキストブロックを画素データのラインに分割する行程と、

非接触で重なっていない文字をラインセグメントから切り出す第1の切出行程と、接触文字を切り出す少なくとも1つの追加の切出行程とを含み、前記分割行程で分割されたラインから文字を切り出す行程と、

前記文字切出行程で切り出された文字を認識し、前記認識に基づいて文字コードを割り付ける行程と、

20 階層的木構造により確定された順に対応して、文字コードをコンピュータのテキストファイルに格納する行程とを備える方法。

【請求項177】 更に、画素画像データが前処理される前処理行程を備える請求項176記載の方法。

【請求項178】 前記前処理行程は画像圧縮行程を含む請求項177記載の方法。

【請求項179】 前記前処理行程は画素画像データの画質を向上する行程を含む請求項178記載の方法。

【請求項180】 更に、文脈チェックを含み、前記認識行程で認識された文字を後処理する行程を備える請求項176記載の方法。

【請求項181】 前記第1の切出行程はラインセグメントをとびとびに処理する行程を含む請求項176記載の方法。

【請求項182】 前記追加の切出行程は、画素画像データの文字について特徴が既知か否かに対応して選択可能である請求項176記載の方法。

【請求項183】 前記追加の切出行程では既知の文字セットの統計に基づいて切り出しを行う請求項182記載の方法。

【請求項184】 前記追加の切出行程では画素密度の回転投影に対応して切り出しを行う請求項182記載の方法。

【請求項185】 画素密度の回転投影は画素密度の垂直投影に基づいて決められた角度で得られる請求項184記載の方法。

【請求項186】 更に、前記第1の切出行程と前記少なくとも1つの追加の切出行程の間に中間の切出行程を備える請求項176記載の方法。

【請求項187】 前記中間の切出行程はラインセグメ

ント内の重なった画素画像の輪郭を追跡する行程を含む請求項186記載の方法。

【請求項188】 更に、文字が間違って切り出された場合に、前記中間の切出行程で切り出された文字を再接続する行程を備える請求項187記載の方法。

【請求項189】 前記認識行程は、前記追加の切出行程の前に前記第1の切出行程で切り出された文字を認識し、前記追加の切出行程は前記認識行程で認識されない文字に対して行なわれる請求項188記載の方法。

【請求項190】 更に、文字が前記認識行程で認識されない場合に、前記少なくとも1つの追加の切出行程で切り出された文字を再接続する行程を備える請求項189記載の方法。

【請求項191】 更に、非テキスト要素の内部で白輪郭を追跡する行程を備える請求項176記載の方法。

【請求項192】 更に、非テキスト要素に識別子を付加する行程を備える請求項191記載の方法。

【請求項193】 非テキスト要素内に含まれる白輪郭の数に基づいて非テキスト要素を表と見なす請求項192記載の方法。

【請求項194】 更に、非格子状配置の白輪郭を再接続する行程を備え、非テキスト要素には前記再接続行程での再接続率に基づいて表識別子が付加される請求項192記載の方法。

【請求項195】 更に、非テキスト要素内の白輪郭の充填率を算出する行程を備え、充填率が高くない場合に非テキスト要素に表識別子を付加する請求項192記載の方法。

【請求項196】 画素画像データでラインの位置を割り出す装置であって、画像データの横方向に伸びた部分の全域で画像密度の水平投影を得る手段と、水平投影に基づいて、画像密度が投影された部分の幅を減少するかどうかを決定する決定行程と、水平投影に基づいて、ラインの位置を割り出す割出行程とを備える装置。

【請求項197】 前記決定手段は、実質的に空白の部分で水平投影を調べる手段と、実質的に空白の部分が見付からない場合に幅を減少する減少手段とを備える請求項196記載の装置。

【請求項198】 前記調べる手段は間隔の小さい非空白の部分に接続する接続手段を含む請求項197記載の装置。

【請求項199】 前記水平投影を得る手段と決定手段とは、順に繰り返して使用される請求項196記載の装置。

【請求項200】 前記割出手段では、水平投影の実質的に空白の領域間の画像データのライン位置を、ラインとして割り出す請求項196記載の装置。

【請求項201】 更に、画素画像データを前処理する

前処理手段を備える請求項196記載の装置。

【請求項202】 段に分離可能な画素画像データ内でラインの位置を割り出す装置であって、画像データの少なくとも1つの横方向に伸びる段の全域で、画像密度の水平投影を得る手段と、前記手段で得られた水平投影に基づいて、画像密度が投影される段の数を増加するかどうかを決定する決定手段と、

前記水平投影に基づいてデジタル画像データ内でラインの位置を割り出す割出手段とを備える装置。

【請求項203】 前記決定手段は、実質的に空白の部分で水平投影を調べる手段と、実質的に空白の部分が見付からない場合に段の数を増加する増加手段とを備える請求項202記載の装置。

【請求項204】 前記調べる手段は間隔の小さい非空白の部分に接続する連結手段を含む請求項203記載の装置。

【請求項205】 前記水平投影を得る手段と決定手段とは、順に繰り返して使用される請求項202記載の装置。

【請求項206】 前記割出手段では、水平投影の実質的に空白の領域間の画像データのライン位置を、ラインとして割り出す請求項202記載の装置。

【請求項207】 更に、画像データの隣接する段のラインが重なるかどうかを決める第2の決定手段を備える請求項202記載の装置。

【請求項208】 前記第2の決定手段は、画像データの隣接する段のブロックが重なるかどうかを決定するスケール不変の決定手段から成る請求項207記載の装置。

【請求項209】 更に、画素画像データを前処理する前処理手段を備え、前記前処理手段は画像縮小手段を含む請求項202記載の装置。

【請求項210】 前記画像縮小は水平方向と垂直方向とで異なる請求項209記載の装置。

【請求項211】 画素画像データのラインセグメントから文字画像を切り出す装置であって、ラインセグメントから非接触で重なっていない文字を切り出す第1の切出手段と、

ラインセグメントから接触文字を切り出す第2の切出手段とを備える装置。

【請求項212】 更に、高解像度の画像セグメントを低解像度に圧縮するように画像データのイメージセグメントを前処理する前処理手段を備える請求項211記載の装置。

【請求項213】 垂直方向と水平方向とで異なる圧縮率が使われる請求項212記載の装置。

【請求項214】 更に、前記第1の切出手段で切り出された文字を認識処理する認識手段を備え、前記第2の切出手段は前記認識手段で認識されなかった文字に対してのみ使用される請求項212記載の装置。

【請求項215】更に、非接触で重なった文字データをラインセグメントから中間で切り出す第3の切出手段を備える請求項214記載の装置。

【請求項216】前記第3の切出手段は、非接触で重なった文字を画素データの輪郭を追跡する輪郭追跡手段を含む請求項215記載の装置。

【請求項217】前記第3の切出手段は、間違って切り出された文字を再接続する再接続手段を含む請求項215記載の装置。

【請求項218】更に、前記認識手段で認識されなかった文字を再接続する第2の再接続手段を備える請求項214記載の装置。

【請求項219】前記第1の切出手段は、前記ラインセグメントを非空白の画素データが見付かるまでとびとびに処理する手段を含む請求項211記載の装置。

【請求項220】更に、前記とびとびの処理手段で非空白の画素が見付かった場合に、緻密に前後をサーチする検索手段を備える請求項219記載の装置。

【請求項221】前記第2の切出手段は非垂直切り出しによって接触文字を切り出す請求項213記載の装置。

【請求項222】非垂直切り出しの角度は画素密度の垂直投影特性に対応して決められる請求項221記載の装置。

【請求項223】更に、垂直投影特性に基づいて少なくとも1つの回転された投影特性を得る手段を備えることを特徴とする請求項222記載の装置。

【請求項224】更に、前記垂直投影特性の角度に近接する角度で複数の回転された投影特性を得る手段を備える請求項223記載の装置。

【請求項225】非垂直切り出しの角度は、前記前記回転投影特性を得る手段と垂直投影特性を得る手段とから得られた投影特性の最小値に基づいて得られる請求項224記載の装置。

【請求項226】前記第2の切出手段は、前記ラインセグメント内の文字の特徴が既知かどうかに基づいて選択可能である請求項211記載の装置。

【請求項227】前記第2の切出手段は、前記文字セット内の文字データのスペースの統計に基づいて切り出す請求項226記載の装置。

【請求項228】更に、小さな文字のセットを再接続する第3の再接続手段を備える請求項227記載の装置。

【請求項229】文字画素データのラインセグメント内で接触文字を切り出す装置であって、ラインセグメント内で画素データの垂直投影特性を得る手段と、

垂直投影特性の最小値から隣接する最大値へ方向から少なくとも1つの角度を算出する第1の算出手段と、

前記第1の算出手段で算出された角度に基づいて、少な

くとも1つの回転された投影特性を算出する第2の算出手段と、

前記回転投影特性の最小値に対応する位置で前記回転投影特性の角度にラインセグメントを切る切断手段とを備える装置。

【請求項230】前記第1の算出手段は、前記垂直投影特性を複数のしきい値と比較して前記最小値と隣接する最大値との位置を決定する比較手段を含む請求項229記載の装置。

【請求項231】更に、最小値と隣接する最大値とが見付からない場合に、前記しきい値を増加させる増加手段を備える請求項230記載の装置。

【請求項232】最小値は、少なくとも一方の側が第2しきい値より大きい最大値によって囲まれ、第1しきい値より小さな垂直投影特性上の位置として割り出される請求項231記載の装置。

【請求項233】更に、各々が前記第2の算出手段で算出された回転投影特性に近接する角度で算出される複数の回転投影特性を算出する第3の算出手段を備える請求項229記載の装置。

【請求項234】更に、複数の回転投影特性と垂直投影特性との全体の最小値を割り出す割出手段を備え、前記切断手段は最小値の位置に対応する位置で最小値が得られた角度に対応する角度に切る請求項233記載の装置。

【請求項235】画素画像データ内で文字を認識する装置であって、

画素画像データから画素ブロックを選別する選別手段と、

30 少なくとも1つの段全体の画素密度の水平投影に基づいてテキストブロックを少なくとも1つの段に適応的に分配することにより、テキストブロックを画素データのラインに分割する分割手段と、

非接触で重なっていない文字がラインセグメントから切り出される第1の切出手段と、接触文字を切り出す少なくとも1つの追加の切出手段とを含み、前記分割手段で分割されたラインセグメントから文字を切り出す切出手段と、

前記切出手段で切り出された文字を認識する認識手段と、

40 前記認識手段で認識された文字を前記選別手段で確定された順に対応して格納する格納手段とを備える装置。

【請求項236】更に、画素画像データが2値の画素画像データでない場合に、画素画像データを2値の画素画像データに変換する変換手段を含む画素画像データの入力手段を備える請求項235記載の装置。

【請求項237】更に、画素画像データが前処理される前処理手段を備える請求項235記載の装置。

【請求項238】前記前処理手段は画像圧縮手段を含む請求項237記載の装置。

【請求項239】 前記前処理手段は画素画像データの画質を向上するフィルタ手段を含む請求項237記載の装置。

【請求項240】 更に、前記認識手段で認識された文字を後処理する後処理手段を備える請求項235記載の装置。

【請求項241】 前記後処理手段は文脈チェック手段を含む請求項240記載の装置。

【請求項242】 前記第1の切出手段はラインセグメントをとびとびに処理する手段を含む請求項235記載の装置。

【請求項243】 前記追加の切出手段は、画素画像データの文字について特徴が既知か否かに対応して選択可能である請求項235記載の装置。

【請求項244】 前記追加の切出手段は既知の文字セットの統計に基づいて切り出しを行う請求項243記載の装置。

【請求項245】 前記追加の切出手段は画素密度の回転投影に対応して切り出しを行う請求項243記載の装置。

【請求項246】 画素密度の回転投影は画素密度の垂直投影に基づいて決められた角度で得られる請求項245記載の装置。

【請求項247】 更に、前記第1の切出手段の使用後で前記少なくとも1つの追加の切出手段の使用前に、中間の階層を切り出す中間の切出手段を備える請求項245記載の装置。

【請求項248】 前記中間の切出手段はラインセグメント内の重なった画素画像の輪郭を追跡する輪郭追跡手段を含む請求項247記載の装置。

【請求項249】 更に、文字が間違って切り出された場合に、前記中間の切出手段で切り出された文字を再接続する再接続手段を備える請求項248記載の装置。

【請求項250】 前記認識手段は、前記追加の切出手段の前に前記第1の切出手段で切り出された文字を認識し、前記追加の切出手段は前記認識手段で認識されない文字に対して行なわれる請求項249記載の装置。

【請求項251】 更に、文字が前記認識手段で認識されない場合に、前記少なくとも1つの追加の切出手段で切り出された文字を再接続する第2の再接続手段を備える請求項250記載の装置。

【請求項252】 画素画像データから文字のテキストファイルを形成する装置であって、画素画像データが2値の画素画像データでない場合に、画素画像データを2値の画素画像データに変換する変換手段を含む画素画像データの入力手段と、画素画像データのブロックがテキストブロックか非テキストブロックかに対応して画素画像データのブロックを選別する選別手段と、

テキスト及び非テキストブロックの階層的木構造を得る

手段と、

少なくとも1つの段の全域の画素密度の水平投影に基づいて、テキストブロックを前記少なくとも1つの段に適応的に分配することにより、テキストブロックを画素画像データのラインに分割する分割手段と、

非接触で重なっていない文字をラインセグメントから切り出す第1の切出手段と、接触文字を切り出す少なくとも1つの追加の切出手段とを含み、前記分割手段で分割されたラインから文字を切り出す切出手段と、

10 前記切出手段で切り出された文字を認識し、前記認識に基づいて文字コードを割り付ける認識手段と、

前記階層的木構造により確立された順に対応して文字コードが格納されるコンピュータのテキストファイルとを備える装置。

【請求項253】 更に、画素画像データが前処理される前処理手段を備える請求項252記載の装置。

【請求項254】 前記前処理手段は画像圧縮手段を含む請求項253記載の装置。

20 【請求項255】 前記前処理手段は画素画像データの画質を向上する画質向上手段を含む請求項253記載の装置。

【請求項256】 更に、文脈チェック手段を含み、前記認識手段で認識された文字を後処理する後処理手段を備える請求項252記載の装置。

【請求項257】 前記第1の切出手段はラインセグメントをとびとびに処理する手段を含む請求項252記載の装置。

【請求項258】 前記追加の切出手段は、画素画像データの文字について特徴が既知か否かに対応して選択可能である請求項252記載の装置。

30 【請求項259】 前記追加の切出手段は既知の文字セットの統計に基づいて切り出しを行う請求項258記載の装置。

【請求項260】 前記追加の切出手段は画素密度の回転投影に対応して切り出しを行う請求項258記載の装置。

【請求項261】 画素密度の回転投影は画素密度の垂直投影に基づいて決められた角度で得られる請求項260記載の装置。

40 【請求項262】 更に、前記第1の切出手段と前記少なくとも1つの追加の切出手段の間に中間の階層を切り出す中間の切出手段を備える請求項252記載の装置。

【請求項263】 前記中間の切出手段はラインセグメント内の重なった画素画像の輪郭を追跡する輪郭追跡手段を含む請求項262記載の装置。

【請求項264】 更に、文字が間違って切り出された場合に、前記中間の切出手段で切り出された文字を再接続する再接続手段を備える請求項263記載の装置。

50 【請求項265】 前記認識手段は、前記追加の切出手段の前に前記第1の切出手段で切り出された文字を認識

し、前記追加の切出手段は前記認識手段で認識されない文字に対して行なわれる請求項264記載の装置。

【請求項266】 更に、文字が前記認識手段で認識されない場合に、前記少なくとも1つの追加の切出手段で切り出された文字を再接続する第2の再接続手段を備える請求項265記載の装置。

【請求項267】 画素画像データから文字のテキストファイルを形成する装置であって、

画素画像データが2値の画素画像データでない場合に、画素画像データを2値の画素画像データに変換する変換手段を含み、画素画像データを入力する入力手段と、

画素画像データ内で連結要素の輪郭を追跡する輪郭追跡手段と、追跡された連結要素のサイズに基づいて追跡された連結要素がテキスト要素を含むか非テキスト要素を含むから決める決定手段と、隣接するテキスト要素の近接関係に基づいてテキスト要素を選択的に横方向に連結してテキストラインを形成する第1の連結手段と、隣接するテキストラインの近接関係とテキストライン間の非テキスト要素の位置とに基づいてテキストラインを選択的に縦方向に連結する第2の連結手段とを含み、画素画像データのブロックを選別する選別手段と、

前記輪郭追跡手段で追跡された連結要素に基づいて、階層の木構造を形成する形成手段と、

少なくとも1つの段の全域での画素密度の水平投影に基づいて、テキストブロックを前記少なくとも1つの段に適応的に分配することによって、テキストブロックを画素データのラインに分割する分割手段と、

非接触で重ならない文字をラインセグメントから切り出す第1の切出手段と、接触文字を切り出す少なくとも1つの追加の切出手段とを含み、前記分割手段で分割されたラインから文字を切り出す切出手段と、

前記文字切出手段で切り出された文字を認識し、前記認識に基づいて文字コードを割り付ける認識手段と、階層の木構造により確定された順に対応して文字コードをコンピュータのテキストファイルに格納する格納手段とを備える装置。

【請求項268】 更に、画素画像データが前処理される前処理手段を備える請求項267記載の装置。

【請求項269】 前記前処理手段は画像圧縮手段を含む請求項268記載の装置。

【請求項270】 前記前処理手段は画素画像データの画質を向上する画質向上手段を含む請求項269記載の装置。

【請求項271】 更に、文脈チェック手段を含み、前記認識手段で認識された文字を後処理する後処理手段を備える請求項267記載の装置。

【請求項272】 前記第1の切出手段はラインセグメントをとびとびに処理する手段を含む請求項267記載の装置。

【請求項273】 前記追加の切出手段は画素画像デー

タの文字について特徴が既知か否かに対応して選択可能である請求項267記載の装置。

【請求項274】 前記追加の切出手段は既知の文字セットの統計に基づいて切り出しを行う請求項273記載の装置。

【請求項275】 前記追加の切出手段は画素密度の回転投影に対応して切り出しを行う請求項273記載の装置。

【請求項276】 画素密度の回転投影は画素密度の垂直投影に基づいて決められた角度で得られる請求項275記載の装置。

【請求項277】 更に、前記第1の切出手段の後で前記少なくとも1つの追加の切出手段の前に中間の階層を切り出す中間の切出手段を備える請求項267記載の装置。

【請求項278】 前記中間の切出手段はラインセグメント内の重なった画素画像の輪郭を追跡する輪郭追跡手段を含む請求項277記載の装置。

【請求項279】 更に、文字が間違っって切り出された場合に、前記中間の切出手段で切り出された文字を再接続する第1の再接続手段を備える請求項278記載の装置。

【請求項280】 前記認識手段は、前記追加の切出手段の前に前記第1の切出手段で切り出された文字を認識し、前記追加の切出手段は前記認識手段で認識されない文字に対して行なわれる請求項279記載の装置。

【請求項281】 更に、文字が前記認識手段で認識されない場合に、前記少なくとも1つの追加の切出手段で切り出された文字を再接続する第2の再接続手段を備える請求項280記載の装置。

【請求項282】 更に、非テキスト要素の内部で白輪郭を追跡する手段を備える請求項267記載の装置。

【請求項283】 更に、非テキスト要素に識別子を付加する付加手段を備える請求項282記載の装置。

【請求項284】 非テキスト要素内に含まれる白輪郭の数に基づいて非テキスト要素を表と見なす請求項283記載の装置。

【請求項285】 更に、非格子状配置の白輪郭を再接続する手段を備え、非テキスト要素には前記再接続手段での再接続率に基づいて表の識別子が付加される請求項283記載の装置。

【請求項286】 更に、非テキスト要素内の白輪郭の充填率を算出する算出手段を備え、充填率が低い場合に非テキスト要素に表の識別子を付加する請求項283記載の装置。

【発明の詳細な説明】

【0001】

【産業上の利用分野】 本発明は、文字認識方法及び装置に関わるものであり、特に認識処理に先立って、画像データに基づいて画像データブロックを分類・選別する方

し、前記追加の切出手段は前記認識手段で認識されない文字に対して行なわれる請求項264記載の装置。

【請求項266】 更に、文字が前記認識手段で認識されない場合に、前記少なくとも1つの追加の切出手段で切り出された文字を再接続する第2の再接続手段を備える請求項265記載の装置。

【請求項267】 画素画像データから文字のテキストファイルを形成する装置であって、

画素画像データが2値の画素画像データでない場合に、画素画像データを2値の画素画像データに変換する変換手段を含み、画素画像データを入力する入力手段と、

画素画像データ内で連結要素の輪郭を追跡する輪郭追跡手段と、追跡された連結要素のサイズに基づいて追跡された連結要素がテキスト要素を含むか非テキスト要素を含むから決める決定手段と、隣接するテキスト要素の近接関係に基づいてテキスト要素を選択的に横方向に連結してテキストラインを形成する第1の連結手段と、隣接するテキストラインの近接関係とテキストライン間の非テキスト要素の位置とに基づいてテキストラインを選択的に縦方向に連結する第2の連結手段とを含み、画素画像データのブロックを選別する選別手段と、

前記輪郭追跡手段で追跡された連結要素に基づいて、階層の木構造を形成する形成手段と、

少なくとも1つの段の全域での画素密度の水平投影に基づいて、テキストブロックを前記少なくとも1つの段に適応的に分配することによって、テキストブロックを画素データのラインに分割する分割手段と、

非接触で重ならない文字をラインセグメントから切り出す第1の切出手段と、接触文字を切り出す少なくとも1つの追加の切出手段とを含み、前記分割手段で分割されたラインから文字を切り出す切出手段と、前記文字切出手段で切り出された文字を認識し、前記認識に基づいて文字コードを割り付ける認識手段と、階層の木構造により確定された順に対応して文字コードをコンピュータのテキストファイルに格納する格納手段とを備える装置。

【請求項268】 更に、画素画像データが前処理される前処理手段を備える請求項267記載の装置。

【請求項269】 前記前処理手段は画像圧縮手段を含む請求項268記載の装置。

【請求項270】 前記前処理手段は画素画像データの画質を向上する画質向上手段を含む請求項269記載の装置。

【請求項271】 更に、文脈チェック手段を含み、前記認識手段で認識された文字を後処理する後処理手段を備える請求項267記載の装置。

【請求項272】 前記第1の切出手段はラインセグメントをとびとびに処理する手段を含む請求項267記載の装置。

【請求項273】 前記追加の切出手段は画素画像デー

タの文字について特徴が既知か否かに対応して選択可能である請求項267記載の装置。

【請求項274】 前記追加の切出手段は既知の文字セットの統計に基づいて切り出しを行う請求項273記載の装置。

【請求項275】 前記追加の切出手段は画素密度の回転投影に対応して切り出しを行う請求項273記載の装置。

【請求項276】 画素密度の回転投影は画素密度の垂直投影に基づいて決められた角度で得られる請求項275記載の装置。

【請求項277】 更に、前記第1の切出手段の後で前記少なくとも1つの追加の切出手段の前に中間の階層を切り出す中間の切出手段を備える請求項267記載の装置。

【請求項278】 前記中間の切出手段はラインセグメント内の重なった画素画像の輪郭を追跡する輪郭追跡手段を含む請求項277記載の装置。

【請求項279】 更に、文字が間違っ

て切り出された場合に、前記中間の切出手段で切り出された文字を再接続する第1の再接続手段を備える請求項278記載の装置。

【請求項280】 前記認識手段は、前記追加の切出手段の前に前記第1の切出手段で切り出された文字を認識し、前記追加の切出手段は前記認識手段で認識されない文字に対して行なわれる請求項279記載の装置。

【請求項281】 更に、文字が前記認識手段で認識されない場合に、前記少なくとも1つの追加の切出手段で切り出された文字を再接続する第2の再接続手段を備える請求項280記載の装置。

【請求項282】 更に、非テキスト要素の内部で白輪郭を追跡する手段を備える請求項267記載の装置。

【請求項283】 更に、非テキスト要素に識別子を付加する付加手段を備える請求項282記載の装置。

【請求項284】 非テキスト要素内に含まれる白輪郭の数に基づいて非テキスト要素を表と見なす請求項283記載の装置。

【請求項285】 更に、非格子状配置の白輪郭を再接続する手段を備え、非テキスト要素には前記再接続手段での再接続率に基づいて表の識別子が付加される請求項283記載の装置。

【請求項286】 更に、非テキスト要素内の白輪郭の充填率を算出する算出手段を備え、充填率が低い場合に非テキスト要素に表の識別子を付加する請求項283記載の装置。

【発明の詳細な説明】

【0001】

【産業上の利用分野】 本発明は、文字認識方法及び装置に関わるものであり、特に認識処理に先立って、画像データに基づいて画像データブロックを分類・選別する方

法や装置に関するものである。例えば、画像データがテキスト画像データであるか、中間調（あるいはグレースケール）画像、線画及びフレームなどのような非テキスト画像であるかに応じて、画像データブロックを選別して分類するものである。また、本発明は、認識処理への準備として、テキストブロックからテキストラインを識別して分割し、次にテキストライン中の個々の文字を識別してライン中の他の文字から切り出す方法や装置にも関するものである。

【0002】尚、本出願は付録のコンピュータプログラム（図35～図247参照）とともに提出される。本明細書の開示の一部には、著作権保護に関係するものが含まれている。著作権者は、特許・商標庁の特許ファイルあるいは記録なので本出願の開示文書がファクシミリ再生されることに対しては異議を唱えないが、そうでない場合にはすべての著作権を留保するものとする。

【0003】

【従来の技術】近年、テキストデータ画像を分析してテキストデータ中の個々の文字を認識し、認識された文字に対応してコンピュータが理解可能な文字コードファイル（図35～図247参照）を生成することが可能になってきた。ここで得られたファイルは、ワードプロセッシングやデータプロセッシングのプログラムで操作を加えることができる。以下「文字認識システム」と呼ぶことにするこのようなシステムの利点は、テキストデータを再タイプしたり、あるいは再入力する必要がないことである。例えば、ファクシミリで送られてきた文書や、マイクロフィルムや複写機で再生された文書を文字認識して、文書中の文字や数字を文字コード（例えばアスキーコードコード）からなるコンピュータテキストファイルとして生成することにより、文書の再タイプや再入力をせずに、文書をワードプロセッシングやデータプロセッシングすることができる。

【0004】文字認識すべき文書にはさまざまな種類の画像が含まれることが多く、すべてが認識可能というわけではない。例えば、テキスト画像データの認識は現在でも可能であるが、非テキスト画像データの認識は不可能である。一般に、文字認識すべき文書には、テキスト画像データブロックと中間調画像、線画及び線などの非テキスト画像データブロックとが含まれる。また、文書には、フレームで囲まれていたり囲まれていなかったりする表や表形式のデータなども含まれる。したがって、文字認識処理に先立って、ブロック中の画像データの種類の応じて文書中の個々のブロックを分類し、画像データからテキストタイプのブロックを選別することが必要となる。

【0005】図32は典型的な文書の1ページの例を示す図である。図32において、文書ページ401は2段組のフォーマットで構成されている。ページには、タイトルに適した大きなフォントサイズのテキスト情報が含

まれるタイトルブロック402、テキストデータラインを含むテキストブロック404、テキストではないグラフィックス画像を含むグラフィックスブロック405、テキストや数字情報の表を含む表ブロック406、小さなサイズのテキストデータでグラフィックスブロックあるいは表ブロックに対する見だしが含まれる見だしブロック407が存在する。各情報ブロックはブロック内の情報の種別に応じて分類され、この分類結果に基づいてブロックが分割される。

【0006】画像データからテキスト型のブロックを検出するため、従来は、画像データ中の黒画素を隣接する1つあるいは複数の白画素にまで水平方向・垂直方向に膨張させて、水平方向・垂直方向に画素画像データを塗りつぶす手法を用いていた。このような塗りつぶし法の欠点としては、適切な塗りつぶしパラメータを選択するために、前もってテキスト型の画像データ中の特徴（例えばフォントサイズなど）を知らなければならないことが挙げられる。また、塗りつぶしパラメータの僅かな変更が、選別結果の大幅な変化につながるという問題もある。さらに、塗りつぶし法では、必ずしも原文書の内部構造が保存されるとは限らない。例えば、塗りつぶし法では、2段組の原文書が1段に塗りつぶされてしまうことがある。このような場合には、テキストデータの格納順序がばらばらになってしまい、原テキストの正確な再生が不可能という問題が生じる。この他、テキスト型のデータが非テキスト型のデータまで塗りつぶして、全領域をテキスト型のデータと誤って解釈してしまう場合も、塗りつぶし法ではしばしば生じる。

【0007】ブロック選別に続いて、文書中の文字ごとに文字認識処理を行い、文字に対応するコンピュータコードを得る。ここで、文字ブロックから個々の文字を得るには、2つの処理を行う。第1段の処理では、タイトルブロック202、テキストブロック204、見だしブロック207などの各テキストブロック中の個々のラインが、テキストブロック中の他のラインから分割される。一般に、ライン分割は、各ブロック中の画素密度を水平投影してライン間のギャップを識別することで行われる。すなわち、図33Aに示すように、テキストブロック404は間にギャップ412を有するテキストラインからなる。画素密度の水平投影414は、ブロック404の各ライン上の黒画素数の総和として求められる。テキストラインは投影414中で密度が非ゼロの領域に対応し、テキストライン間のギャップは投影414中の密度がゼロの領域に対応する。したがって、投影密度に基づいて、テキストラインを各ラインごとに分割することができる。

【0008】第2段の処理では、分割されたテキストライン中の個々の文字がテキストライン中の他の文字から切り出される。すなわち、図34Aに示すように、テキストライン411には個々の文字が含まれる。テキスト

法や装置に関するものである。例えば、画像データがテキスト画像データであるか、中間調（あるいはグレースケール）画像、線画及びフレームなどのような非テキスト画像であるかに応じて、画像データブロックを選別して分類するものである。また、本発明は、認識処理への準備として、テキストブロックからテキストラインを識別して分割し、次にテキストライン中の個々の文字を識別してライン中の他の文字から切り出す方法や装置にも関するものである。

【0002】尚、本出願は付録のコンピュータプログラム（図35～図247参照）とともに提出される。本明細書の開示の一部には、著作権保護に関係するものが含まれている。著作権者は、特許・商標庁の特許ファイルあるいは記録なので本出願の開示文書がファクシミリ再生されることに対しては異議を唱えないが、そうでない場合にはすべての著作権を留保するものとする。

【0003】

【従来の技術】近年、テキストデータ画像を分析してテキストデータ中の個々の文字を認識し、認識された文字に対応してコンピュータが理解可能な文字コードファイル（20）を生成することが可能になってきた。ここで得られたファイルは、ワードプロセッシングやデータプロセッシングのプログラムで操作を加えることができる。以下「文字認識システム」と呼ぶことにするこのようなシステムの利点は、テキストデータを再タイプしたり、あるいは再入力する必要がないことである。例えば、ファクシミリで送られてきた文書や、マイクロフィルムや複写機で再生された文書を文字認識して、文書中の文字や数字を文字コード（例えばアスキーコードコード）からなるコンピュータテキストファイルとして生成することにより、文書の再タイプや再入力をせずに、文書をワードプロセッシングやデータプロセッシングすることができる。

【0004】文字認識すべき文書にはさまざまな種類の画像が含まれることが多く、すべてが認識可能というわけではない。例えば、テキスト画像データの認識は現在でも可能であるが、非テキスト画像データの認識は不可能である。一般に、文字認識すべき文書には、テキスト画像データブロックと中間調画像、線画及び線などの非（40）テキスト画像データブロックとが含まれる。また、文書には、フレームで囲まれていたり囲まれていなかったりする表や表形式のデータなども含まれる。したがって、文字認識処理に先立って、ブロック中の画像データの種類の応じて文書中の個々のブロックを分類し、画像データからテキストタイプのブロックを選別することが必要となる。

【0005】図32は典型的な文書の1ページの例を示す図である。図32において、文書ページ401は2段組のフォーマットで構成されている。ページには、タイトルに適した大きなフォントサイズのテキスト情報が含（50）

まれるタイトルブロック402、テキストデータラインを含むテキストブロック404、テキストではないグラフィックス画像を含むグラフィックスブロック405、テキストや数字情報の表を含む表ブロック406、小さなサイズのテキストデータでグラフィックスブロックあるいは表ブロックに対する見だしが含まれる見だしブロック407が存在する。各情報ブロックはブロック内の情報の種別に応じて分類され、この分類結果に基づいてブロックが分割される。

【0006】画像データからテキスト型のブロックを検出するため、従来は、画像データ中の黒画素を隣接する1つあるいは複数の白画素にまで水平方向・垂直方向に膨張させて、水平方向・垂直方向に画素画像データを塗りつぶす手法を用いていた。このような塗りつぶし法の欠点としては、適切な塗りつぶしパラメータを選択するために、前もってテキスト型の画像データ中の特徴（例えばフォントサイズなど）を知らなければならないことが挙げられる。また、塗りつぶしパラメータの僅かな変更が、選別結果の大幅な変化につながるという問題もある。さらに、塗りつぶし法では、必ずしも原文書の内部構造が保存されるとは限らない。例えば、塗りつぶし法では、2段組の原文書が1段に塗りつぶされてしまうことがある。このような場合には、テキストデータの格納順序がばらばらになってしまい、原テキストの正確な再生が不可能という問題が生じる。この他、テキスト型のデータが非テキスト型のデータまで塗りつぶして、全領域をテキスト型のデータと誤って解釈してしまう場合も、塗りつぶし法ではしばしば生じる。

【0007】ブロック選別に続いて、文書中の文字ごとに文字認識処理を行い、文字に対応するコンピュータコードを得る。ここで、文字ブロックから個々の文字を得るには、2つの処理を行う。第1段の処理では、タイトルブロック202、テキストブロック204、見だしブロック207などの各テキストブロック中の個々のラインが、テキストブロック中の他のラインから分割される。一般に、ライン分割は、各ブロック中の画素密度を水平投影してライン間のギャップを識別することで行われる。すなわち、図33Aに示すように、テキストブロック404は間にギャップ412を有するテキストラインからなる。画素密度の水平投影414は、ブロック404の各ライン上の黒画素数の総和として求められる。テキストラインは投影414中で密度が非ゼロの領域に対応し、テキストライン間のギャップは投影414中の密度がゼロの領域に対応する。したがって、投影密度に基づいて、テキストラインを各ラインごとに分割することができる。

【0008】第2段の処理では、分割されたテキストライン中の個々の文字がテキストライン中の他の文字から切り出される。すなわち、図34Aに示すように、テキストライン411には個々の文字が含まれる。テキスト

ラインにおいて各文字を他の文字から切り出すために、ラインセグメント411の各列ごとに黒画素数の垂直方向の総和を求め、画素密度の垂直投影416を得る。文字415は投影416中で密度が非ゼロの領域に対応し、文字間のギャップは投影416中で密度がゼロの領域に対応する。こうして、個々の文字がラインセグメント中の他の文字から切り出される。

【0009】このような処理においてはいくつかの問題点が存在する。例えば、文書が斜めに画像スキャナに入力され、図33Bに示すように傾斜角 θ で画像メモリに格納される場合が多いが、このような場合には、第1ライン418のテキストが第2ライン419のテキストとライン420で重なってしまうため、必ずしもライン分割が可能であるとは限らない。すなわち、画素密度の水平投影421は非ゼロ値のみとなり、ゼロ値が存在しないためライン間のギャップの検出が不可能になる。

【0010】このような問題点に鑑みて、テキストブロック404を図33Cに示すような複数の列422、424に分割して、各列ごとに独立に水平投影を得る手法が用いられている。すなわち、図33Cに示すように、水平投影422aは列422に対応し、水平投影424aは列424に対応するものである。各列でテキストラインが重なっていないければ、各列中のテキストラインの検出が可能となる。図33Cでは2列に分割する例を示しているが、一般には5列から10列に分割し、テストで最大傾斜角 θ sまでの傾きを有していても、ブロック中の他のラインから個々のラインの分割が可能であるようにしている。

【0011】しかし、各列ごとに水平画素投影を得なければならない、またこのようにして得た各水平画素投影も個々に処理しなければならないため、ライン分割処理がきわめて時間のかかる処理となる。また、最大傾斜角 θ sの文書にまで対応しなければならないため、傾斜角が小さくて1列あるいは数列への分割で十分である多くの文書に対しても、すべての列を処理しなければならない、この点でも時間のかかる処理となっている。上記のような2段階処理のうち1つの問題は、個々の文字をラインセグメント中の他の文字から切り出す第2段の処理において生じる。

【0012】すなわち、文字間に垂直な空間が存在する場合には、図34Aに示したような処理で良好な結果が得られるが、文字が垂直方向に重なっていた場合や2つあるいは複数の文字が接触している場合には、文字分割処理を適切に行うことができない。イタリックフォントの場合や、複数回の複写処理やファクシミリ送信によって画品質が劣化している場合など、このような状況に陥る可能性は高い。例えば、図34Bに示すように、イタリックテキストでは単語「Satisfy」中の文字「f」と「y」とが垂直方向に重なっており、画素密度の垂直投影425が文字間でゼロにならないことがあ

る。そのため、文字「f」と「y」とを分離することはできない。また、文字「t」と「i」も接触しているため、同様にこれら2つの文字の分離も不可能である。

【0013】

【発明が解決しようとしている課題】本発明は、前記従来の欠点を除去し、高速かつ正確に文書上の文字を認識してテキストファイルを作成する文字認識方法及び装置を提供する。また、高速かつ正確に文書上のテキストと非テキストとを選別して、テキストブロックを割り出す方法及び装置を提供する。

【0014】また、高速かつ正確に文書上の非テキストを分類する方法及び装置を提供する。また、高速かつ正確にテキストブロックからテキストラインを分割する方法及び装置を提供する。また、高速かつ正確に傾いた文書のテキストブロックからテキストラインを分割する方法及び装置を提供する。

【0015】また、高速かつ正確にテキストラインから文字を切り出す方法及び装置を提供する。また、不適切に切り出された文字の再切り出しが可能な方法及び装置を提供する。

【0016】

【課題を解決するための手段】この課題を解決するために、本発明の文字認識方法及び装置は以下の構成を含む。画素画像データが入力され、2値の画素画像データでない場合は2値の画素画像データに変換される。

【0017】画素画像データのブロックが、画素画像データ内で連結要素の輪郭を追跡し、連結要素がテキスト要素を含むか非テキスト要素を含むかを連結要素のサイズに基づいて決定し、隣接するテキスト要素の近接状態に基づいてテキスト要素を選択的に連結してテキストラインを形成し、隣接するテキストラインの近接状態とテキストライン間の非テキスト要素の位置とに基づいてテキストラインを選択的に連結することにより、選別される。階層的木構造が連結要素に基づいて形成される。

【0018】テキストブロックは、段の全域での画素密度の水平投影に基づいてテキストブロックを少なくとも1つのラインに分割することで、テキストラインに分割され、非接触で重なっていない文字を切り出す第1の切り出しと接触文字間を切り離す第2の切り出しとの2回の切り出しで、文字が分割されたラインから切り出される。切り出された文字は認識され、認識に基づいて文字コードが割り当てられる。文字コードは階層的木構造による確定された順でコンピュータのテキストファイル内に格納される。

【0019】必要ならば、非テキスト要素は階層的木構造により確定された順に対応して格納された文字コードの間に散在してもよい。画素画像データは例えば画像圧縮や画質向上等の前処理をされ、認識された文字は例えば文脈チェック等の後処理をされてもよい。非テキスト要素の特徴に基づいて、非テキスト要素に識別子が付与

される。例えば、非テキスト要素の内部で白輪郭の追跡が行なわれ、非格子状配置の白輪郭が再連結されて、白輪郭の充填率が計算されて、白輪郭の数、非格子状配置の白輪郭の再連結率、あるいは白輪郭の充填率等に基づいて、非テキスト要素には表の識別子が付与される。

【0020】非接触で重なっていない文字の切り出しは、ラインセグメントへのとびとびの処理で達成され、接触文字の切り離しは、接触文字間の空間に関する情報が既知か否かに対応して達成される。空間に関する情報が既知の場合は、空間の統計値に基づいて切り出しが達成される。空間に関する情報が無い場合、接触文字間の切り離しは、回転された投影によって決められた角度と位置で接触文字を斜めに切り離すように、画素密度の回転投影に対応して達成される。不適切に切り出された文字は再接続される。

【0021】

【実施例】本発明は、複写機、ファクシミリ、ビデオカメラあるいはスチルビデオカメラ、レーザビームプリンタなどの画像処理装置あるいは画像再生装置などのような、文字認識処理が望まれるさまざまな装置において実施可能である。このような装置においては、文字画像を有する画像を、文字画像が文字認識されるように処理あるいは再生する。この際、認識した文字画像を標準文字セットあるいはフォントに替えて、原文字画像ではなく標準文字を再送あるいは再生することもできる。また、本発明は、汎用コンピュータや、パーソナルコンピュータ、ワードプロセッシングあるいはデータプロセッシング機器などのオフィス機器や、複数のビジネス機器を単一の統合パッケージにまとめた統合オフィス自動機器などにおいても実施可能である。

【0022】図1は、スキャン、ファクシミリ、情報送受信、情報処理（ワードプロセッシングやデータプロセッシングなどの処理を含む）などの機能を有する統合オフィス自動機器として、本発明の一実施例を示すブロック図である。図1に示す装置において、画像の入力は、ファクシミリ送信による入力、原文書のスキャンによる入力、モデムを介して離れた地点からの入力などで行われる。本実施例によれば、画像中の文字を認識し、認識文字のコンピュータテキストファイルを生成し、装置内のワードプロセッシング、スプレッドシート処理ならびに他の情報処理機能を利用してテキストファイルの修正を行うことができる。この処理に続いて、修正されたテキストファイル（あるいは無修正のテキストファイル）の再送や、音声合成技術を用いてテキストファイルのスピーカや通常の音声電話への「話しかけ」などによる出力が行われる。

【0023】図1において、プログラム可能なマイクロプロセッサなどの中央処理ユニット（CPU）10はバス11に接続されている。また、画像を画素ごとにスキャンして画像メモリ（例えば、以下で示すRAM20）

に蓄積するスキャナ12、デジタルデータを電話線15aを通してアナログ形式で送受信するモデム14、画像データを電話線15bを通して送受信するファクシミリ16（不図示の電話を含むこともある）なども、バス11に接続される。なお、電話線15aと15bとは同一の線であって、不図示のネットワーク制御ユニットで制御されるものでも良い。さらに、バス11には、CPU10によって実行される1つあるいは複数のコンピュータプログラムを蓄えるリードオンリメモリ（ROM）17、認識処理において入力文字と比較する文字の辞書を保持している文字辞書19、入力画像データ、処理画像データならびに画像の構造などの情報を蓄えるランダムアクセスメモリ（RAM）20、文字認識処理において文字の認識結果を出力する出力装置21（ディスクメモリ、スピーカあるいは音声電話線インタフェースを有する音声合成装置など）、装置によって処理された画像を表示するプリンタ/ディスプレイ22、オペレータが装置を操作するためのキーボード24なども接続される。

【0024】なお、ここでは、バス11に接続された装置が協調して統合オフィス自動機器を構成しているが、これらの装置のうちのいくつかあるいはすべてが単独で用いられることも可能なことは明らかであろう。スキャナ12、モデム14及びファクシミリ16は、画像データを装置に入力するための種々の入力手段である。スキャナ12では、原画像はラインごと画素ごとにスキャンされ、CPU10の制御のもとで画像データの画素はRAM20中の画像メモリにビットマップ形式で蓄えられる。モデム14では、電話線15aを介してアナログ形式で受信した画像データがデジタル画素形式に変換され、デジタル画素データはRAM20中の画像メモリに蓄えられる。ファクシミリ16では、電話線15bを介して修正ハフマンランレンクス符号などの符号形式で受信した圧縮画像データを、ファクシミリ16において従来手法でデジタル画像の画素データに復号して、CPU10によって画像データの画素はRAM20中の画像メモリにビットマップ形式で蓄えられる。ここで、他の画像入力手段を用いることはもちろん可能で、ディスクメモリなどの大容量の蓄積メディアから画像データを取り出したり、ビデオカメラあるいはスチルビデオカメラから入力することもできる。

【0025】ファクシミリ16や出力装置21は、文字認識された画像データを装置から出力するための種々の手段である。ファクシミリ16では、本実施例に基づき認識処理された文字画像が標準文字セットあるいはフォントに変換され、装置から送信される。これにより、例えば、文字画像を含む画像を受信して、文字画像を文字認識し、再送に先立って認識文字を標準文字フォントに変換することで、劣化画像の品質の改善を図ることができる。

【0026】モデム14や出力装置21は、画像データ

で認識された文字をASCIIコードなどで出力あるいは蓄積するための種々の手段である。すなわち、文字は装置（ディスクメモリなど）に蓄えられたり、モデム14を介して離れた地点に転送するために出力される。この際、ASCIIコードをファクシミリ互換のフォーマットに変換するなどの文字変換手段を設ければ、ファクシミリ16を用いることなくモデム14を介して離れた地点のファクシミリ機に転送を行うことができる。

【0027】プリンタ/ディスプレイ22は、本実施例に基づく文字認識処理の流れの監視、ならびに文字認識の各ステップの永久記録の出力および表示を行うための手段である。キーボード24は、図1の装置の動作に対するオペレータの制御を可能にしている。図2は、本実施例の文字認識処理のフローチャートである。図2に示した処理ステップは、プログラムROM17に蓄えられているコンピュータプログラムに基づいてCPU10で実行される。

【0028】ステップS201において、画素単位の画像データ（以下画素画像データ）は装置に入力され、RAM17に蓄えられる。画像データは画像を画素ごとに表現したものである。この際、画素データは2値画素データ、すなわち黒ならびに白画像データであることが好ましい。なお、画像データは、複数のグレイ階調値の1つで各画素が表現される中間調画像であっても、画素色を表す複数ビットワードで各画素が表現されるカラー画像データであっても良い。このような場合、すなわち画素データが2値画素データでない場合には、非2値画素データを2値画素データに変換するためのしきい値処理をRAM20に蓄積する前に行う必要がある。

【0029】ステップS201で入力された画素画像データは、左上隅から右下隅に読み進められるようなポートレイト画像であることが好ましい。画像がポートレイト画像でない場合、例えばランドスケープ画像である場合には、画素画像データをポートレイト型の画像に変換する必要がある。この変換処理は、オペレータがキーボード24から画像データの変換を指示することで行うことができる。

【0030】ステップS202では、画像データの前処理が行われる。一般に、前処理フィルターを用いて画像データの画質を向上し、劣化した文字や画像を改善する。画質向上手法として適切な手法は、1991年10月4日出願の米国出願番号07/771,220に示されている。尚、ステップS202では、精度の劣化は避けられないが、画素画像データ中の複数の画素を削除したり圧縮したりすることで、認識処理の高速化を図ることも可能である。例えば、 $m \times n$ 画素ブロックの画素の平均値を求め、この平均値で $m \times n$ 画素ブロックを代表させることも可能である。

【0031】ステップS203では、ブロック選択を行って各セグメント画像中の画像データの型を調べ、プロ

ック中の情報がテキスト情報か、グラフィックス情報か、線画情報か、画像情報かなどを識別する。また、ブロック選択ステップS203では、画像中の各部分を以下のステップS212で示すような適切な順序で再生することができるように、画像の階層的木構造をも生成する。この階層的木構造には、例えば、2段組の画像データにおいて第1段のテキストから第2段のテキストに飛んで読み進めることのないように、第1段のテキストの後に第2段のテキストを再生できるような情報が含まれる。なお、ステップS203におけるブロック選択処理については、以下でより詳細に説明する。

【0032】RAM20に蓄えられた画像から第1の情報ブロックが選択され、ブロック選択ステップS203で判別されたブロック識別子に応じて、ステップS204において選択されたブロックがテキストブロックであるかどうか調べられる。第1ブロックがテキストブロックでない場合には、処理をステップS205に進め、次のブロックを選択してステップS204に戻る。

【0033】ステップS204においてブロックがテキストブロックである場合には、処理をステップS206に進め、テキストブロックに対してライン分割処理を行う。ライン分割処理では、テキストブロック中の個々のテキストラインをテキストブロック中の他のテキストラインから分離して、以下に詳細に説明するように分離したラインごとに順々に処理を行う。ステップS207では、ライン中の各文字をライン中の他の文字から切り出し、以下に詳細に説明するように個々の文字に対して文字認識処理を行う。

【0034】ステップS208では、文字ごとに各文字の認識処理が行われ、既知の手法で文字辞書19に蓄えられている標準文字と各文字との比較処理が行われる。文字辞書19中の索引は一般には1つの文字に対応するが、分離が難しいような文字列（例えば、「f1」）や分離されやすい1文字（例えば「j」）に対しては他の索引が用意されることもある。すなわち、分離が難しいような接触文字ペアを辞書の索引とすることも、分離されやすいような1文字中の部分を辞書の索引とすることもある。比較処理に基づいて文字画像に対する識別子が選択され、選択された文字識別子はRAM20に蓄えられたり、出力装置21に出力される。また、識別された文字をプリンタ/ディスプレイ22に表示することもできる。

【0035】ステップS209では、テキストブロックに対する処理が終了したかどうか調べられる。処理が終了していなければ、処理をステップS206（あるいは適宜ステップS207）に戻し、ライン分割処理、文字切り出し処理及び文字認識処理とを繰り返す。ブロック処理が終了していれば、処理をステップS210に進め、1ページ分の処理を終了したかどうかを調べる。1ページ分の処理が終了していなければ、処理をス

31

ステップS205に戻し、ページ中の次ブロックを選択して処理を続ける。

【0036】1ページ分の処理が終了していれば、処理をステップS210からステップS211に進め、後処理を実行する。後処理では、文脈チェックやスペルチェックなどの処理を行い、文字認識ステップS208で認識された文字識別子を、ステップS208のような個々の文字ごとではなく文字の周囲の文脈に基づいて、全体的な見地から修正する処理を行う。ステップS212では、ブロック選択ステップS203で定義された階層的木構造に基づいて画像データが再生される。ページ再生では画像データが適切な順序で配置される。例えば、脚注はメインテキストから分離され、段は他の段と混ざることなく順番に配置され、グラフィックスや線画はページ中の認識文字テキストの適当な位置に挿入される。このようなグラフィックス画像や線画の見だしも図に隣接して挿入される。この際、他の規則を用いることもでき、例えばページの物理的再生ではなく、ページからテキストのみを抽出するという規則を用いることもできる。なお、このようなページ再生の規則は、装置の情報処理機能を用いてユーザが定義できる。

【0037】ステップS213では、再生されたページが出力装置21などに蓄えられる。ここでは、ROM17に蓄えられており、CPU10で実行される他の応用プログラムを用いて、スプレッドシートやワードプロセッシング処理などの情報処理を行うことができる。そして、処理された情報（あるいは適宜処理していない情報）は、ファクシミリ16やモデム14などを用いて、またはコンピュータテキストファイルを「しゃべる」音声合成装置を用いた通常の音声電話を介してなどのさまざまな手段を用いて再送される。

【0038】図3は、図2のステップS203のブロック選別処理を説明するための一般化した処理流れ図である。ここで、ブロック選別処理は文字認識システムとは別に利用可能であることに注意されたい。すなわち、画像再生装置においてブロック選別処理を用いて、ある種のブロックに対しては画像再生の第1の手法を施し、別の種類のブロックに対しては画像再生の第2の手法を施すといったことも可能である。

【0039】ブロック選別の処理速度の向上を望む場合には、ステップS300で画像データを縮小する処理を行う。画像データの縮小処理を行った場合には、ブロック選別処理は縮小画像に対して行われる。なお、この際、図2の残りの文字認識処理（ステップS204からステップS213）に影響を及ぼさないように、ブロック選別処理終了時には、縮小前の画像データに対して選択ブロックが割り当てられる。画像データ縮小処理は、 $m \times m$ 画素ブロック中の黒画素の連結性を調べながら行われる。例えば、 3×3 画素ブロック中に連結された黒画素が2個ある場合には、 3×3 画素ブロックを1つの

32

黒画素に縮小する。逆に、 3×3 画素ブロック中に連結された黒画素が2個ない場合には、 3×3 ブロックを1つの白画素に縮小する。

【0040】ステップS301では、画素画像を解析して連結された要素を検出し、連結要素をサイズや他の連結要素との相対的な位置に応じて分類する処理を行う。ここで、連結要素は白画素に完全に囲まれた黒画素のグループである。すなわち、連結要素は、少なくとも1つの白画素で他の黒画素グループと完全に分離されている黒画素グループである。図4を参照して以下で詳細に説明するように、ステップS301では、連結要素を検出して、連結要素から得たサイズ情報やいくつかの統計的な値に基づいて各連結要素の分類を行う。

【0041】まず、以下で詳細に説明するように、各連結要素をテキスト要素か非テキスト要素かに分類する。さらに、非テキスト部については、フレームデータであるか、中間調画像であるか、線画であるか、表あるいはテキストデータを表のように構成したものであるか、あるいは未知の要素で分類不可能であるかなどを解析処理して決定する。また、各連結要素ごとに階層的木構造を構成して、連結要素の構造的な記述データを求め、ステップS212で説明したようにデジタルデータの再生を容易にする。

【0042】ステップS302では、水平方向に近い位置関係にあり、ギャップラインマーカを横切らないような連結要素をラインとしてグループ化する。この際、ステップS301で作成した木構造を用いて、テキスト要素と非テキスト要素とが不適切にグループ化されないようにする。また、ステップS302では、段間で垂直方向に伸びるギャップや非テキスト要素の垂直方向に伸びる境界を検出して、テキスト要素を段ごとにまとめる。ここで得られた段構造は階層的木構造に組み込まれ、適宜木構造の更新が行われる。

【0043】ステップS303では、ステップS302でグループ化されたラインの内、垂直方向に近接するラインを垂直方向にグループ化して、ブロックを形成する。ここで、非テキスト要素は垂直方向にソートされ、画像ページの境界として用いられる。なお、2つの非テキスト要素の間に位置するテキストライン要素は、他のテキストライン要素とは別に処理される。また、ステップS303では、ステップS301で分類できなかった非テキスト要素が、大きなフォントサイズのタイトルであるかどうかを判別する。タイトルであると判別されると、その部分に「タイトル」属性を付与し、木構造を更新する。ここで得られたタイトルは、ステップS212に基づくページの再生の際に有用な情報となりうる。

【0044】図4A～図4Cは、画素画像データの連結要素をどのように検出して、どのようにこれらの連結要素を分類するかを示す詳細なフローチャートである。図4A～図4Cに示される処理ステップは、ROM17に

保持されているプログラムステップに基づいてCPU10で実行される。ステップS401では、輪郭追跡を行い画素画像データの連結要素を検出する。輪郭追跡処理は図5Aに示すように画像データをスキャンすることで行われる。スキャン処理は矢印Aで示されるように画像の右下部から左に行われ、画像の右側の端に達したときに上に進むように行われる。なお、この際、逆方向すなわち左上から右下にスキャンすることもできる。スキャン処理中に黒画素を検出すると、31で示すような星形パターンの順序で隣接画素を調べ、黒画素の隣接画素もまた黒画素であるかどうかを判別する。星形パターン31には8個の番号付けされたベクトルが中心点から伸びているため、このような輪郭追跡を以下「8方向」追跡と呼ぶことにする。黒画素が隣接して存在する場合には、画像の外側輪郭を追跡し終えるまで上述の処理を繰り返す。

【0045】すなわち、図5Bに示すように、矢印A方向へのスキャン処理によって、まず文字「Q」の端に対応する点32が検出される。次いで、星形パターン31に基づいて隣接画素が判別され、文字「Q」の外側輪郭が追跡される。この際、閉輪郭の内部の輪郭追跡処理は行わない。連結要素を1つ検出して、8方向追跡で輪郭の追跡を終えると、次の黒画素を検出するまでスキャン処理を続ける。すなわち、完全な黒領域である物体34の8方向追跡を行う。また、手書き単語「non-text」の非テキスト物体35を、単語「word」を構成する個々の文字のテキスト物体36中の個々の文字と同様に追跡する。図5Aに示すスキャン処理は、画素データ中のすべての連結要素を検出し、8方向追跡して輪郭を検出するまで行われる。

【0046】次いで処理をステップS402に進め、各連結要素を矩形で囲む処理を行う。具体的には、各連結要素の周りを包囲する最も小さな矩形を描く。すなわち、図5Bに示すように、物体32の周りには矩形37が、物体34の周りには矩形39が、物体35の周りには矩形40が、テキスト物体36a、36b、36c、36dの周りには矩形41a、41b、41c、41dが描かれる。

【0047】ステップS403では、各矩形要素に対して木構造中の位置が割り当てられる。ほとんどの場合、ステップS403で得られる木構造では、画素画像中の各物体は木の根に直接つながる。これは、連結要素の外側輪郭のみを追跡しており、閉輪郭の内部は追跡していないためである。すなわち、図5Cに示すように、連結要素32に対応する矩形37はページの根に直接つながる。これに対し、非テキスト物体35に対応する矩形40や、テキスト物体36a、36bに対応する矩形41a、41bなどのように、矩形が他の矩形の内部に完全に位置するような連結要素は、囲っている連結要素（この場合には要素34）の子ノードとして位置付けられ

る。また、要素34などのように少なくとも1つの子を有する連結要素は、それ自身を「主な子ノード」として位置付ける。すなわち、図5Cに示すように、要素39が「主な子ノード」として、要素39の他の子ノード40、41a、41bとともに位置付けられる。

【0048】ステップS404では、木構造の第1レベルの各連結要素をテキスト要素あるいは非テキスト要素に分類する。この分類処理は2つのステップで行われる。第1ステップでは、連結要素の矩形と所定のしきい値サイズとの比較を行う。連結要素を囲む矩形の高さが、想定される最大フォントサイズに対応する所定の第1しきい値以上であれば、あるいは連結要素を囲む矩形の幅が、ページ幅を経験的に決められた一定値で割った値以上であれば（「5」で良好な結果が得られることがわかっている）、連結要素を非テキスト要素と分類し、要素に「非テキスト」の属性を付与する。

【0049】第2ステップでは、残りすべての要素すなわち非テキストと分類されなかった要素と、残りすべての連結要素のサイズの集合に基づいて適当に求められるしきい値との比較を行う。具体的には、非テキスト要素として分類されなかったすべての矩形の平均の高さを求め、この平均高にスカラー値を掛け合わせたものを（「2」を用いると好都合である）適当に求められるしきい値とする。そして、適当に求めたしきい値以上に大きな要素はすべて非テキストであると考え、非テキスト要素として分類する。また、適当に求めたしきい値以下の小さな要素はすべてテキスト要素であると考え、このようにして要素の分類を行い、適切な属性が付与される。

【0050】なお、図4A～図4Cの残りを参照して以下で詳細に説明するように、ここで求めた分類結果は以下で改善される。木構造の第1レベルの各要素をテキスト要素あるいは非テキスト要素として分類すると、テキスト要素の主な子ノードを含むすべての子ノードをテキスト要素として分類する。これに対して、非テキスト要素の主な子ノードは非テキスト要素として分類するが、非テキスト要素の他のすべての子ノードはテキスト要素と分類する。

【0051】ステップS405において、最初の要素を選択する。要素がテキスト要素であれば（ステップS406）、ステップS407に処理を進め次の要素を選択する。非テキスト要素を選択して処理をステップS408に進めるまで、ステップS406とS407の処理を繰り返す。ステップS408では、非テキスト要素に子ノードが存在するかどうか調べられる。例えば、図5Cに示されるように、非テキスト要素39には、非テキストの主な子ノード39とテキストの子ノード40、41a、41bとが存在する。

【0052】ステップS408において子ノードが存在する場合には、処理をステップS409に進め、要素に

フィルタ処理が施され、要素が中間調（あるいはグレイ階調）要素であるかどうか判別される。中間調フィルタ処理では要素の子ノードを調べ、要素の子ノードのうち「雑音サイズ」以下であるようなサイズのものを求める。ここで「雑音サイズ」の要素とは、高さが画像データにおいて想定される最小のフォントサイズ以下であるような要素のことである。雑音サイズ以下のサイズの子ノード数が全体の子ノード数の半分以上であれば、要素は中間調画像として判断される。そこで、ステップS410で処理をステップS411に進め、「中間調」属性を要素に付与する。

【0053】次いでステップS412で、中間調画像中のテキスト要素をチェックする。具体的には、木構造中の中間調画像の子ノードのうちテキストサイズの子ノードを修正して、テキストサイズの要素を中間調画像の子ノードとしてではなく、中間調画像と同一レベルのノードに配置する。これにより、中間調画像中のテキストサイズの要素に対しても、適切だとすれば文字認識処理を行うことができる。この処理を終えろと処理をステップS407に戻し、次の要素を選択して処理を行う。

【0054】ステップS409の中間調フィルタ処理において要素が中間調画像でないと判別された場合には、処理をステップS410からステップS413に進め、さらなる処理を施すために要素の主な子ノードを選別する。そして、処理をステップS414に進める。ステップS408において非テキスト要素が子ノードをもたないと判別される場合、あるいはステップS413においてさらなる処理を施すために主な子ノードが選択されると、ステップS414で当該要素に対してフレームフィルタ処理が施される。フレームフィルタ処理は、当該要素がフレームであるかどうかを調べ、当該要素を囲むような矩形とはほぼ同一の幅/高さを有する平行水平線と平行垂直線とを検出するための処理である。具体的には、画面中の行ごとに当該要素中の連結要素内部の最大長を求めて、連結要素が調べられる。

【0055】すなわち、図6Aに示されるように、非テキスト要素42には連結要素43が含まれるが、その輪郭は8方向追跡によって44で示すように追跡処理される。行「1」における連結要素内部の最大長は、輪郭の左端45aから右端45bまでの距離 x_1 となる。また、行「J」では、連結要素内部の長さとして、連結要素の境界上の点46aと46bの距離、ならびに点47aと47bの距離との2つが存在する。ここで、点46aと46bの距離の方が点47aと47bの距離よりも長いため、距離 x_1 が行「J」における連結要素内部の最大長となる。

【0056】非テキスト要素42中のn個の行ごとに距離「x」を求め、以下の不等式を満たすかどうかを調べ、非テキスト要素がフレームであるかどうかを判別する。

【0057】

【数1】

$$\sum_{k=1}^N \frac{(X_k - W)^2}{N} < \text{threshold (しきい値)}$$

ここで、 x_k は（上述のように）連結要素内部のk行目の最大長、Wは矩形要素42の幅、Nは行数、しきい値はフレームが画像データ中で傾いていてもフレーム検出が可能となるように前もって計算した値である。ここで、1度の傾き角を許容するには、しきい値として $\sin(1^\circ)$ をL倍したものに、ステップS404で計算したテキストの平均の高さをオフセットとして加えたものを用いると良い。

【0058】上記の不等式が満たされれば、要素はフレーム要素であると判断され、ステップS415からステップS416に処理を進め、「フレーム」属性を要素に付与する。ここで、複数の属性が各要素に付与されることに注意されたい。すなわち、「フレーム-表」、「フレーム-中間調」などのようにフレームの属性が付与されることもある。

【0059】ステップS416に続いて、フレーム要素中に表データあるいは表のように組織されたデータが存在するかどうかを調べる処理を行う。そこで、ステップS417において、連結要素の内部を調べて白輪郭を求める。白輪郭は、黒画素ではなく白画素に着目するという点を除けば、上のステップS401で検出した輪郭と同一のものである。すなわち、図7Aに示されるように、非テキスト要素の内部を非テキスト要素の右下から左上へ矢印Bの方向に沿ってスキャンする。このスキャン処理によって第1の白画素が検出されると、星形パターン51の順序で白画素の隣接要素がチェックされる。星形パターン51には1から4までの数が付与されたベクトルがある。このため、このような処理に基づく白輪郭追跡処理のことを以下「4方向」白画素追跡と称する。

【0060】黒画素に囲まれた白輪郭すべてを追跡し終えるまで、4方向の白画素追跡処理を繰り返す。例えば、白輪郭追跡処理によって、黒画素セグメント52、53、54、55ならびに内部に存在する56のような他の黒画素で構成される内部輪郭の画素が追跡される。このようにして白輪郭を位置付けると、矢印B方向へのスキャン処理を、非テキスト物体に囲まれたすべての白輪郭を追跡するまで繰り返す。

【0061】ステップS418では、非テキスト要素の密度を計算する。密度は連結要素中の黒画素数を計数して、黒画素数を矩形中の全画素数で割ることで計算される。ステップS419では、非テキスト要素中で検出された白輪郭数の数をチェックする。白輪郭の数が4以上であれば、非テキスト画像が表またはテキストブロックを表のように並べたものである可能性が高い。

37

【0062】そこで、ステップS420において、白輪郭の充填率を求める。白輪郭の充填率は、白輪郭が非テキスト画像中の領域を占める割合のことである。例えば、図7Aに示されるように、白輪郭充填率には、57や59のような斜線領域で完全に空の白領域と、60や61のような領域で中に黒画素を含むような白領域とが含まれる。この充填率が高いと、非テキスト画像が表またはテキストデータを表のように並べたものである可能性が高い。そこで、ステップS421において充填率を
10 チェックする。この際、高い充填率であれば、非テキスト画像は表またはテキストデータを表のように並べたものであると考えられる。

【0063】この処理の信頼性を向上させるために、白輪郭が水平ならびに垂直方向に格子状の構造をなしているかどうかを調べる。すなわち、ステップS422において、少なくとも2つの白輪郭の境界線が水平方向ならびに垂直方向に一致しないような非格子状の白輪郭を再接続する。例えば、図7Aに示されるように、白輪郭59の左境界線62及び右境界線63は、白輪郭60の左境界線64及び右境界線65と垂直方向に一致する。そのため、これらの白輪郭は格子状に配置されており、白輪郭の再接続は行わない。同様に、白輪郭59の上部境界線66及び下部境界線67は、白輪郭70の上部境界線68及び下部境界線69と水平方向に一致する。そのため、これらの白輪郭は格子状に配置されており、これらの白輪郭の再接続は行わない。
20

【0064】図7Aから図7Dまでは、白輪郭を再接続する場合を説明するための図である。図7Bは非テキスト要素71を示しており、これはステップS201で示した中間調画像の2値画像へのしきい値処理などによっ
30 て得られるものである。非テキスト画像71には、黒領域72と白領域74、75、76、77、78、79が存在する。ここで、これらの白領域の充填率は十分高く、ステップS421において処理は再接続ステップS422に進められたものとする。

【0065】まず、図7Cに示されるように、白輪郭75の左境界線及び右境界線と、白輪郭76の左境界線及び右境界線とを比較する。すると、これらの境界線は一致しないため、白輪郭75と白輪郭76とは再接続されて図7Cの接続白輪郭76'が生成される。次いで、図7Dに示されるように、白輪郭77の上部境界線及び下部境界線と、白輪郭79の上部境界線及び下部境界線とを比較する。すると、これらの境界線は一致しないため、白輪郭77と白輪郭79とは1つの接続白輪郭77'として結合される。再接続される輪郭がなくなるまで、これらの処理を水平方向に垂直方向に繰り返す。

【0066】すなわち、上述のように、表をなす白輪郭は再接続されにくいのに対し、中間調画像や線画などの表でないような白輪郭は再接続されやすい。そこで、ステップS423において接続率を調べる。ここで、再接
50

38

続率が高ければ、あるいは再接続処理後に残った白輪郭の数が4より少なければ、処理をステップS428に進め、以下で詳述するように、非テキスト要素に「中間調画像」もしくは「線画」の属性を付与する。ステップS423において再接続率が高くなく、且つ少なくとも4つの白輪郭が残っていれば、処理をステップS424に進め、非テキスト画像に「表」の属性を付与する。

【0067】ステップS425では、新たに属性が付与された表の内部を調べ、8方向追跡処理で連結要素を検出し分類する。ステップS426では、この新たな内部の連結要素に基づいて階層的木構造を更新する処理を行う。続くステップS427では、ステップS402からステップS404で示したような手法でもって、内部連結要素をテキスト要素もしくは非テキスト要素に再分類し、適当な属性を付与する。これらの処理を終えると、処理をステップS407に戻し次の要素が選択される。

【0068】ステップS421とS423に戻って、ステップS421で充填率が高くない場合、あるいはステップS423で再接続率が高い場合には、非テキストフレーム要素は中間調画像もしくは線画である可能性が高いと考えられる。ここで、要素を中間調画像として分類するか、あるいは線画として分類するかの決定は、要素中の黒画素の平均水平ラン長、要素中の白画素の平均水平ラン長、白画素と黒画素との比率、及び密度に基づいて行われる。一般に、暗めの画像は中間調画像であると考えられ、明るめの画像は線画であると考えられる。

【0069】具体的には、白画素の平均水平ラン長がほぼゼロ（すなわち、全体が暗めあるいはまだらな画像）であって、ステップS418で求めた密度によって要素が白っぽいというよりもむしろ黒っぽい場合（すなわち、密度が1/2程度の第1しきい値より高い）には、フレーム要素は中間調画像であると判別される。密度が第1しきい値以下である場合には、要素は線画として判別される。

【0070】白画素の平均水平ラン長がほぼゼロではなく、黒画素の平均水平ラン長より長い場合には、フレーム要素は線画であると判別される。しかし、白画素の平均水平ラン長が黒画素の平均ラン長以下（すなわち、暗めの画像）である場合には、さらなる判別処理が必要となる。具体的には、黒画素数が白画素数よりかなり少ない（すなわち、黒画素数を白画素数で割った値が約2の第2しきい値より大である）場合には、フレーム要素を中間調要素として判別する。これに対し、黒画素数を白画素数で割った値が第2しきい値以下であっても、ステップS418で求めた密度が第1しきい値より高ければ、フレーム要素を中間調画像として判別する。これ以外であれば、フレーム要素は線画として判別される。

【0071】ステップS428においてフレーム要素が線画として判別されれば、処理をステップS429に進めて「線画」属性を付与し、ステップS430ですべて

の子ノードを削除する。なお、要素が線画として判別されると、線画要素のいかなるブロックも文字認識のために選択されることはない。そして、処理をステップS407に戻し次の要素を選択する。

【0072】一方、ステップS428においてフレーム要素が線画として判別されなかった場合には、処理をステップS431に進めて「中間調」属性を付与し、ステップS432でフレーム中間調要素のテキストサイズの子ノードを削除する。ここで、テキストサイズは、ステップ404において述べたように平均の要素の高さとして求められる。また、テキストサイズより大きなすべての子ノードは、フレーム中間調要素の子ノードのままとする。これらの処理を終えると、処理をステップS407に戻し次の要素を選択する。

【0073】ステップS419に戻り、白輪郭の数が4より小さい場合には、フレーム要素は表であるとは判別されない。そこで、処理をステップS433に進め、ステップS418で求めた密度と約0.5のしきい値とを比較する。ここで、しきい値は、フレーム内のテキスト要素や線画が占める面積は画素の半分以下であることを鑑みて選択されている。

【0074】密度がしきい値より小さければ、処理をステップS434に進め、上述のようにフレーム要素の内部構造を作成する。すなわち、処理をステップ401に戻し、フレーム要素の内部構造に対する処理を行う。ステップS433において、密度が所定のしきい値以上であった場合には、処理をステップS442に進め、フレーム要素を線画、中間調画像あるいは分類不可能（すなわち、フレームが「未知」）のどれかに分類する。

【0075】ステップS415に戻り、ステップS414におけるフレームフィルタ処理で非テキスト要素中にフレームが検出されなかった場合には、処理をステップS435に進め、非テキスト要素中に線が含まれるかどうかを調べる。線は、テキスト境界を明確にするのに適した非テキスト要素である。この際、このような線に囲まれたテキストは線のそばに位置することが多いため、テキストが線に接するということもありうる。そのため、テキストが接していても接していなくても線を検出できるような線の検出処理を行う。

【0076】テキストが接していない線を検出するには、非テキスト要素の長さ方向のヒストグラムを求める。すると、図6Bに示されるように、線のヒストグラム48はほぼ均一な分布となり、ヒストグラムの高さは線幅にほぼ等しくなる。また、線幅は、非テキスト要素の幅「W」にほぼ等しい。ここで、線幅と非テキスト要素の幅との差は、画素画像を生成する際の原文書の傾き角 θ によるものである。そこで、非テキスト要素が線を含むかどうかを調べるために、ヒストグラム中の各セルkの高さ49を非テキスト要素の幅Wと比較する。すなわち、これらの値間の実効値誤差としきい値とを次式

のように比較する。

【0077】

【数2】

$$\sum_{k=1}^N \frac{(\text{cell}_k - W)^2}{N} < \text{threshold} \quad (\text{しきい値})$$

ここで、しきい値は、非テキスト要素中の線の傾き角 θ を許容できるように設定される。例えば1°の傾き角であれば、しきい値として

【0078】

【数3】

$$\sum_{k=1}^N \left[\frac{k \sin(1^\circ)}{N} \right]^2$$

を用いると良好な結果が得られる。上の不等式によってテキスト要素が接していない線が検出されなかった場合には、テキスト要素が接している線を含む要素であるかどうかを調べる処理を行う。テキスト要素に接している線が非テキスト要素中に含まれるかどうかを判別するために、要素の境界に沿って線が長く伸びているかどうかを調べる。すなわち、要素を通して線が長く伸びていれば、図6Cに示すように要素を囲む矩形の境界は線にきわめて近いところに位置する。そこで、矩形の境界近くの第1番目の黒画素位置の均一性を、境界からの距離の2乗の和を求めることで判断する。すなわち、次式の不等式を満たすかどうかを調べる（図6C参照）。

【0079】

【数4】

$$\sum_{k=1}^N \frac{X_k^2}{N} < \text{threshold} \quad (\text{しきい値})$$

ここで、2乗の和が所定のしきい値より小である場合には、テキスト要素が接している線要素と判別する。この際、しきい値として、テキスト要素が接していない線の検出におけるしきい値を用いると、良好な結果が得られる。ステップS435において線を検出すると、処理をステップS436からステップS449に進め、「線」属性を非テキスト要素に付与する。そして、処理をステップS407に戻し、次の要素を選択する。

【0080】一方、ステップS435で線が検出されなかった場合には、処理をステップS436からステップS437に進め、非テキスト要素のサイズを調べる。ここで、サイズが所定のしきい値以下であると、非テキスト要素の分類が不可能となる。なお、このしきい値は最大のフォントサイズに基づいて決められるものであり、最大フォントサイズの半分の値で良好な結果が得られる。そして、処理をステップS438に進めて「未知」属性を非テキスト要素に付与し、処理をステップS407に戻して次の要素を選択する。

【0081】ステップS437においてサイズが所定の

しきい値よりも大であった場合には、処理をステップS 4 3 9、S 4 4 0、S 4 4 1に進め、ステップS 4 1 7、S 4 1 8、S 4 1 9で述べたように、非テキスト要素内部の白輪郭を追跡し、非テキスト要素の密度を求め、白輪郭の数を調べる。ステップS 4 4 1において白輪郭の数が4以下であれば処理をステップS 4 4 2に進め、要素サイズを計算し、要素が線画や中間調画像であるのに十分な大きさであるかを調べる。このサイズ計算は、非テキスト要素の高さと幅ならびに黒画素の最大ラン長に基づいて行われる。具体的には、非テキスト要素の高さや幅が最大フォントサイズ以下であれば、非テキスト要素は中間調画像や線画であるほど大きくはないと判断し、処理をステップS 4 4 3に進め「未知」属性を付与する。また、非テキスト要素の幅は最大フォントサイズより大であるが、黒画素の最大ラン長は最大フォントサイズ以下である場合にも、処理をステップS 4 4 3に進め、「未知」属性を付与する。そのあと、ステップS 4 0 7に処理を戻し、新たな要素を選択する。

【0082】ステップS 4 4 2において非テキスト要素が線画や中間調画像であるのに十分な大きさであると判断されると、ステップS 4 4 4に処理を進め、非テキスト要素が線画であるか中間調画像であるかの判別を行う。ステップS 4 4 4からステップS 4 4 8までの処理は、それぞれステップS 4 2 8からステップS 4 3 2までの処理と同一のものであり、ここでの説明は省略する。

【0083】図4A～図4C（図3中のステップS 3 0 1）を参照して以上説明したように、画素画像中のすべての連結要素を検出して分類すると、図14に示されるような木構造が得られる。図14に示されるように、木構造の根は画素画像データのページに対応する。根からの子ノードには、テキストブロックや、未知、フレーム、絵、線などからなる非テキストブロックが存在する。また、フレームの子ノードには、テキストブロック、未知の非テキストブロック、テキストブロックを有する表、絵、線などが存在する。

【0084】図10は画素画像データの典型的なページ90を示したものであり、大きなフォントサイズのテキスト91、テキスト93などを含む表92、テキストデータ94、水平線95、もう一つのタイトル96、2段落のテキストデータ97及び見だし99を備えるフレーム線画98とタイトル100で始まり、テキストデータ101、見だし103を備えるフレーム中間調画像102、テキストデータ104、水平線105、テキストデータの最終段落106へと続く第2段とが示されている。図11は、ステップS 3 0 1に基づく処理後の同一画素画像を示したものである。図11に示されているように、画素画像データ90中の連結要素が矩形ブロックで囲まれており、矩形ブロックの内部についてはステップS 4 1 5からステップS 4 3 4までのフレーム処理で

判別される。

【0085】ステップS 3 0 2では、ステップS 3 0 1で得られたすべてのテキスト要素を、木構造の位置に関わらず水平方向にグループ化する。グループ処理は各テキスト要素の密集性ならびにその近傍関係とに基づいて行われる。なお、この際、段に対応する垂直方向に伸びたギャップが検出され保持される。以下、図8を参照して、ステップS 3 0 2で行う詳細な処理について説明を加える。図8に示される処理ステップは、ROM 17に保持されるプログラムステップに基づいてCPU 10で実行される。

【0086】ステップS 8 0 1では、非テキスト要素の左端ならびに右端から垂直にギャップラインマーカを伸ばす処理を行う。すなわち、図11に示されるように、ギャップラインマーカ109aと109bとをテキストあるいは非テキスト要素（ここでは要素95）に達するまで垂直に伸ばす。また、ギャップラインマーカ109cと109dとをテキストあるいは非テキスト要素（ここでは要素95）に達するまで垂直方向に伸ばす。同様に、残りの非テキスト要素の左端ならびに右端からギャップラインマーカを垂直に伸ばす。このようなギャップラインマーカを用いることで、画素画像データにおいて段に対応するギャップ位置の判別が容易になる。

【0087】ステップS 8 0 2において、図11のテキスト要素107のようなテキスト要素は、連結によりギャップラインマーカを横切らず、且つ他のテキスト要素と接しているかあるいは他のテキスト要素から所定のしきい値内の距離にあれば、1つのテキストラインに連結される。なお、適切なしきい値として、ステップS 4 0 4で求めた平均テキスト長に経験的に得られたスカラー値を掛けたものを用いる（「1. 2」を用いると良好な結果が得られる）。また、連結処理を行うに先立って、テキスト要素間の垂直ギャップを調べ、段構造を示唆するような垂直ギャップが存在するかどうかを判別する。すなわち、図11に示されるように、ギャップ108はテキスト要素のベアの間に存在するが、このギャップはテキスト画像データ中で垂直方向に数ライン程度の長さとなるため、テキスト要素が他の要素から所定のしきい値内の距離に位置しているにも関わらず、ステップS 8 0 2ではギャップが維持される。

【0088】ステップS 8 0 3では、連結ステップS 8 0 2で連結されなかったテキスト要素のベアが連結によりギャップラインマーカを横切らず、隣のラインの第3テキスト要素と共に重なるような場合に、これらのテキスト要素のベアを連結する。このようなステップにより、段構造を示すギャップではなく単にテキストライン中のランダムな空間構成に起因するようなギャップを効果的に除去することが可能となる。例えば、図11において、ステップS 8 0 2ではギャップ108の連結は行われぬが、ギャップ両側のテキスト要素は1行下のラ

43

インの第3テキスト要素と重なり、またギャップラインマーカを横切らないため、ステップS803においてこのギャップが除去される。

【0089】ステップS804では、これらの処理結果に基づいて木構造の適切な更新が行われる。図12は、ステップS302のグループ化処理の結果を示しており、図15はステップS302のグループ化処理で修正された木構造を示している。図12に示されるように、各々が接しているテキスト要素は、ライン110のようにテキストラインにグループ化される。すなわち、木構造中に位置するテキスト要素をテキストラインに連結する処理を行うが、111などのようにテキスト要素が木構造中のフレーム-表ノードの下に位置する場合にも連結処理が行われる。ここで、このようなグループ化処理は、上のステップS417からS439で求めた白輪郭境界を横切ることはなく、表中の個々の項目が1つの連続するテキストラインにグループ化されることはない。また、左段と右段との間のギャップは維持される。さらに、非テキスト要素は再連結されない。すなわち、112や113などの要素のように、互いに所定のしきい値内の距離にあっても、非テキスト要素のグループ化は行わない。

【0090】図15には、新たなグループ化処理の結果に基づいて修正した木構造が示されている。図8（図3中のステップS302）を参照して説明したようにテキスト要素のテキストラインのグループ化を行った後には、ステップS303に示されるようにテキストラインを垂直方向にグループ化してテキストブロックを生成する処理を行う。

【0091】以下、図9を参照してこの処理について詳述する。グループ化処理は、テキストライン要素の密集性や非テキスト要素の位置に基づいて行われる。例えば、間に存在する非テキストラインは境界を示すため、これを利用すれば非テキストライン両側のテキストラインを1つのテキストブロックにグループ化することを行われる。なお、処理は、2つの連続する非テキストライン要素間のすべてのテキストラインに対して1度に行われる。また、ステップS303では、テキスト要素を非テキスト要素に連結すべきか（例えば、非テキスト画像とそのテキスト見だし）、非テキスト要素を他の非テキスト要素に連結すべきか（例えば、中間調画像と線画）の判断も行う。

【0092】図9は、テキストラインのテキストブロックへのグループ化を示す詳細なフローチャートである。ステップS901においては、最大の予想フォントサイズより小ではあったが平均テキストサイズより大であったため、ステップS404で非テキスト要素に分類された非テキスト要素からタイトルブロックの形成を行う。隣り合う非テキスト要素のうち同じようなサイズの要素に対してはすべてグループ化を行い、タイトルブロック

44

を形成し、「タイトル」属性をこのグループに付与する。ここでグループ化されなかった残りすべての非テキスト要素に対しては「絵-テキスト」属性が付与される。また、それに応じて木構造の更新も行う。ここで得られたタイトルは、ページ再生時（ステップS212）に有用な情報となる。

【0093】ステップS902では、2つのテキストラインにまたがる非テキスト要素の位置を明確にする。このような非テキスト要素はテキストブロック間の境界となり、テキストラインを1つのテキストブロックにグループ化してしまうことを避けることができる。ステップS903では、2つのステップでテキストラインの垂直方向のグループ化を行い、テキストブロックを形成する。第1のステップでは、画素密度の垂直ヒストグラムを求めるなどして、段間のギャップを検出する。第2のステップでは、垂直方向に連続するテキストライン間の垂直距離がステップS404で求めたテキストの高さより小さければ、それぞれの段ごとにテキストラインのグループ化を行う。このステップS903の処理により、図2のライン114のような同一テキスト段落中のテキストラインを、テキストブロックにグループ化する。

【0094】ステップS904では、垂直ならびに水平方向に隣接するテキストブロックのグループ化処理を行う。ここで、これらのテキストブロックが非テキスト要素で分離されていず、またこれらのブロックを連結してもステップS903のヒストグラムから求めたギャップが保持されるときに、テキストブロックのグループ化が行われる。また、ブロック間の距離がステップS404の垂直の高さに基づいて計算される所定のしきい値より小であるときに、テキストブロックはグループ化される。このステップS904の処理により、図12中の段落115のラインと段落116のラインのテキストブロックはグループ化されるが、段落117と118のラインのテキストブロックはそれらの間に非テキスト要素119（線）を有するためグループ化が行われない。

【0095】ステップS905では、テキストブロックを非テキストブロックに連結すべきか、非テキストブロックを他の非テキストブロックに連結すべきかを判断する。ここで、テキストブロックの非テキスト-タイトルブロック、非テキスト-中間調ブロック、非テキスト-ライン接触ブロックとの連結は、以下のように行われる。

(1) テキストブロックが非テキスト-タイトルブロックと水平方向に近い位置にあり、垂直方向に重なっている場合には、テキストブロックを非テキスト-タイトルブロックに連結する。

(2) テキストブロックがワードサイズのブロックより小さくて（水平、垂直方向とも）、テキストブロックの隣にワードサイズのテキストブロックが存在しないときには、テキストブロックを非テキスト-中間調画像ブロ

ックの中に位置付ける。

(3) テキストブロックが非テキストライン接触ブロックと重なっていれば、ライン接触ブロックはテキストの下線である可能性が高いため、ライン接触ブロックをテキストブロックに変換する。

*

	中間調	線画	テキスト-絵	タイトル
中間調	テスト1	非連結	常に連結	非連結
テキスト-絵	テスト1	テスト2	テスト2	テスト3
線画	テスト1	テスト1	非連結	非連結
タイトル	非連結	非連結	非連結	テスト3

この表中のテストは以下の通りである。

テスト1：1つのブロックが完全に他のブロック内に位置すれば連結。

テスト2：絵-テキストの幅がワードサイズのブロックの幅より小さければ連結。

テスト3：ブロックが近接していれば連結。

【0098】ステップS906では、適切な属性が付与され、上述の処理結果に基づいて木構造の更新が行われる。図13は図9の処理を行った結果のブロック構造であり、図16はその木構造の例である。図13において、ブロックには、タイトルブロック120、テキストブロック121及び絵データ122が含まれる。また、フォームデータも含まれ、123は表構成のデータを有するフレーム要素を示しており、124はテキスト要素125を有するフレーム要素を示している。なお、非テキストライン画像127は図13に示されるさまざまな要素を分離している。

【0099】図3から図16を参照しながら説明したブロック選択処理を終えると、上述のように文字認識処理は図2のステップS204に処理を進める。すなわち、階層的木構造中の第1ブロックを選択して認識処理を行う。このブロックがテキストブロックでない場合には、処理をステップS204からステップS205に進め、木構造中の次のブロックを選択する。テキストブロックが選択されるまでステップS204とS205を繰り返し、テキストブロックが選択された時点でステップS2※

*【0096】また、以下の表にしたがって、非テキストブロックは他の非テキストブロックと連結される。

【0097】

【表1】

※06に処理を進め、ラインの分割処理を行う。

【0100】図17は、図2のライン分割ステップS206で実行される処理ステップを詳細に示すフローチャートである。図17に示される処理ステップは、プログラムROM17に保持されるコンピュータプログラムにしたがってCPU10で実行される。ステップS1701に先立って画像縮小処理を行っても良い。しかし、ライン分割処理や文字分割処理は水平方向の空白により影響を受けやすいため、画像縮小処理を行うにあたっては分割精度に影響を与えないような注意が必要である。すなわち、水平方向と垂直方向とでそれぞれ異なる画像縮小手法を用いることが好ましい。垂直方向では画素の結合を「OR」論理で行い、垂直方向の対象画素のうち1つでも黒画素が存在すれば黒画素が出力される。すなわち、垂直方向の2：1の画像縮小処理では、2つの垂直画素のいずれかが黒画素であれば黒画素が出力される。これに対して水平方向では画素の結合を「AND」論理で行い、水平方向の対象画素すべてが黒画素であれば黒画素が出力される。すなわち、水平方向の3：1の画像縮小処理では、3つの画素がすべて黒画素のときのみ黒画素が出力される。

【0101】垂直方向に3：1の縮小を行い、水平方向に2：1の縮小を行う場合の処理例を以下に示す（「0」は白画素を「X」は黒画素を表す）。

【0102】

【表2】

原画像	垂直縮小 (OR)	水平縮小 (AND)
X O X O X X X O O O O O O X O O O X X X O O X O	X X X O O X X X	X O O X

画像縮小処理を終えると、この縮小画像に対してライン分割処理と文字切り出し処理とが行われる。ここで、図2の残りの認識処理（すなわち、ステップS208からステップS213まで）に影響を及ぼさないように、ラ

イン分割処理と文字切り出し処理とを終えた時点で、文字間の切り出しは縮小していないもとの画像データに対して行う。

【0103】ステップS1701では、ステップS20

4で選択されたテキストデータブロックごとに画素密度の水平投影を求める。画素密度の水平投影は、画素画像の各行ごとに黒画素数を計数して得られる。ここで、画素密度の水平投影は全体のテキストブロックに対して求めることが好ましいが、これは本質的な点ではない。すなわち、テキストブロックを複数の画素列、例えば2あるいは3列に分割して、各列ごとに画素密度の水平投影を求めることができる。もちろん、このような処理では、本発明の処理時間短縮という利点は失われる。

【0104】ステップS1702では、水平投影のうちゼロでない領域を求め、どれかが最大フォントサイズと等しい所定のしきい値より大であるか否かを調べる。最大フォントサイズ以下であれば、水平投影はページ上のラインを均一に分割していることになり、処理をステップS1703に進める。ステップS1703では、画素密度の水平投影のうち近接している領域を連結する。この処理を図18A～図18Dを用いて説明する。図18Aは典型的なテキストブロック230を示しており、文字画像のライン231と233、ならびに雑音スポット232（すなわち、文字情報ではない黒画素）が含まれる。これに対応する画素密度の水平投影は234に示される。水平投影234に示されるように、領域235は文字「1」の上の点に対応し、領域236はライン231上の残りの文字に対応し、領域237と238は雑音スポット232に対応し、領域239はライン233上の文字に対応する。これらの各ピークにより、境界が水平投影のゼロ値となる領域が定義される。ステップS1703では、同一テキストライン上に存在するピーク235と236のような近接領域は連結され、テキストラインとは関係のないピーク237と238のような近接領域は連結されないことが望まれる。

【0105】ステップS1703で述べたように、近接領域を連結するためには、画素密度の水平投影を投影の上部から下部まで（テキストブロックの上部から下部まで）調べる。第1の領域を検出すると、その下の次の領域の水平投影をスキャンして、2つの領域の高さを比較する。上の領域の最も高い要素の高さが下の領域の高さより小さく、2つの領域の間隔が上の領域中の最も高い要素の高さより小さければ、これら2つの領域を連結する。ここで、この近接領域を連結する処理は、スケールに対して不変であることに注意されたい。すなわち、テキストの近接領域を連結する際には、テキストサイズが12ポイントであるとか8ポイントであるなどの知識は不要である。

【0106】そこで、図18Aに戻り、領域235の高さと領域236の高さとの比較を行うと、上の領域235の高さが下の領域の高さより小さいことがわかる。また、2つの領域間のギャップは領域235の高さより小さいと判断される。したがって、図18Bに示されるように、領域235と236とは1つの領域236'とし

て連結される。

【0107】画素密度の水平投影を下方向に調べると、領域237が検出される。この場合、領域236'の高さが領域237の高さ以上であるため、これら2つの領域は連結されない。さらに、画素密度の水平投影を下方向に調べると、領域237の高さが領域238の高さ以下であり、また領域間のギャップは領域237の高さ以下であることがわかる。そこで、図18Cに示されるように、領域237と238とは1つの領域238'として連結される。画素密度の水平投影をさらに下方向に調べると、領域239が検出される。この場合、領域238'の高さは領域239の高さより低いが、領域間のギャップは領域238'を構成している領域237と238とのどちらかの高さより大きいことがわかる。そこで、これら2つの領域は連結されない。

【0108】また、近接領域の連結処理のあとに各領域の高さを調べて各領域の高さがラインの最小の高さに対応するしきい値より大であるかどうかを判断することもできる。この際、しきい値は、これまでに検出された領域の高さの平均として適当に設定される。領域がしきい値以下の高さの場合には、領域を画素データ中の雑音スポットに起因するものと判断し削除することができる。したがって、領域238'の高さは、領域236'、238'、239の高さの平均として決められるしきい値以下であるため、図18Dに示されるように領域238'は削除される。

【0109】これらの処理を終えると、図17に戻って処理をステップS1704に進め、領域を個々のラインセグメントに分割し、図2のステップ207で示される文字切り出しに処理を進める。ステップS1702において、ステップS1701で処理された領域が大きすぎる領域であった場合には、テキストラインが傾いていると判断される。例えば、図19Aに示されているように、テキストブロック240には複数の傾いたテキストライン241が含まれる。ステップS1702に基づく処理では、244に代表テキストを示したようにテキストラインが水平方向に相互に重なりあってしまうため、242のような画素密度の水平投影が得られる。

【0110】そこで、処理をステップS1705に進め、テキストブロックを段に分割する。図19Bに示されるように、テキストブロックの段数を2倍にする。すなわち、テキストブロック240を2つの段に分割する。この際、少なくとも1つの共通の画素が重なり合うように段を分割することが好ましい。また、テキストブロックを2ブロック以上、例えば3あるいは4ブロックに分割することも可能である。

【0111】ステップS1706では図19Bの247や249のように各段ごとに画素密度の水平投影を求め、ステップS1707で領域が大きすぎるかどうかを再び調べる。領域が大きすぎる場合には、処理をステッ

ブS1708に進め、段数を再び増加させる。例えば、図19Cに示されているように、段数を更に2倍にする。また、ステップS1709において、段の幅が最低限度より大であるかどうかを確認する。この最低限度は、これ以上段数を増加させると適切なライン分割が不可能になる点を示している。好適な実施例では、最低限度は16画素幅である。ステップS1709において最低限度に達していると判断されると、ライン分割は不可能であると表示して処理を終了する。一方、最低限度まで達していない場合には、処理をステップS1706に戻し、新たな段ごとに水平投影を再び計算する。ステップS1707において領域が大きすぎないと判断された場合には、処理をステップS1710に進める。すなわち、図19Cに示されるように、ラインセグメント以下の幅の領域が割り出された。そこで、処理をステップS1710に進め、ステップS1703で説明したように近接領域を連結する。そして、ステップS1711において、各段ごとに単一ラインセグメントに対応する領域を割り出す。

【0112】すなわち、図19Cにおいて、単一ラインセグメントに対応する領域250、251、252、253を検出する。そして、領域が異なる段間で接触しており単一ラインセグメントを構成するかどうかを判別するために、各段を上から下に調べ各段の第1領域を検出する。そこで、その領域に接触している領域を調べ、図19Dに示されているように2つの距離を求める。この2つの距離、(1)2つの領域を合せたときの全長距離Aと、(2)2つの領域の共有領域すなわち2つの領域の交差領域の距離Bとの2つである。そして、比A/Bを求め、2つの領域が多くの部分で重なっているかを確かめるためにしきい値と比較する(しきい値として5を用いると良好な結果が得られる)。比A/Bがしきい値より小であれば、2つのブロックは多くの部分で重複していることになり、ブロックは単一ラインセグメントを構成していると考えられる。そこで、ステップS1712において、比A/Bで求められる接触領域を単一ラインセグメントとして割り出される。

【0113】ここで、比A/Bの計算と、比A/Bとしきい値との比較処理とは、スケールに不変な処理であり、ライン中のテキストサイズに関わらず、重なった接触領域は単一ラインセグメントとして割り出される。このようなスケール不変の性質は、ラインセグメント中のテキストサイズが既知である必要がないという点で望ましいものである。

【0114】図17で述べたラインの分割処理を終えると、図2のステップS207で説明し、詳細が図20に示される文字分割(文字切り出し)に処理を進める。図20に示されるように、文字切り出しは多階層の処理で行われ、各階層は徐々に複雑な文字切り出し処理を行う。すなわち、文字切り出し処理は3つの処理に分類さ

れる。相互に接触ならびに重なっていない文字間の切り出し、相互に接触してはいないが重なっている文字間の切り出し、接触している文字間の切り出しの3つである。例えば、図34Bに示されるように、文字「S」と「a」は接触ならびに重なっていないため第1の処理に分類される。一方、文字「f」と「y」は接触してはいないが重なっているため第2の処理に分類される。さらに、文字「t」と「l」は相互に接触しているため第3の処理に分類される。

【0115】図20に示されるように、各階層は3つの処理のうちの1つを実行するように構成される。すなわち、階層1(261)では接触ならびに重なっていない文字間の切り出しが行われる。階層1に続いて、当該テキストの性質及び特徴に関する知識の有無に応じて処理を進める。テキストが単一スペースのテキストであれば、すなわち文字が垂直で文字間が均一であれば(例えば「クーリエ」フォント)、処理を階層2(262)に進め、接触している文字の切り出しを行う。単一スペースの文字であっても、コピーやファクシミリ転送の繰り返しに起因する画像の劣化のため、文字が相互に接触することがある。そして、文字認識処理263に処理を進め、図2のステップS209に進む。

【0116】一方、テキストブロックの性質及び特徴に関して何の情報もない場合、あるいはテキストが単一スペースのテキストでなければ、処理を階層2(264)に進め、接触はしていないが重なっている文字間の切り出しを行う。そして、階層1と階層2との切り出しで得られた文字に対して265で認識処理を行う。認識処理265で認識されなかったすべての文字に対しては、階層3(266)の処理を行う。すなわち、文字が認識されない理由として、文字の切り出しが完全に行われないため実際には認識不能な文字は接触している複数の文字と考えられる。そこで、階層3では接触している文字の切り出しが行われる。階層3で文字を切り出すと、267で文字認識処理を行う。認識処理が成功した場合には、処理を図2のステップS209に戻す。一方、認識処理が再び失敗した場合には、階層3での切り出しが不適切であったと判断する。そこで、認識不能であった切り出しを再接続し、再び階層3の切り出し処理と認識処理とを行う。このような処理を、文字の切り出しが不可能になるまで繰り返し行う。

【0117】「単一スペース(mono-spaced)」262か「全スペース(all-spacing)」264かのどちらに処理を進めるかは、オペレータからの入力によって選択される。オペレータからの入力がない場合には、デフォルトとして処理を「全スペース」264に進める。というのは、この階層は単一スペースのみならず非単一スペース文字にも適用できるためである。

【0118】図21から28までは階層1から階層3までの処理を説明する図であり、図30と図31は269

で示される再接続処理を説明する図である。図21は非接触で重なっていない文字を切り出す階層1の切り出し処理を説明するフローチャートである。階層1では、2つの文字間の白画素すなわち空白を検出することにより、非接触で重なっていない文字の切り出しを行う。

【0119】具体的には、図21のステップS2101に示されるように、空白でない画素すなわち黒画素が検出されるまでラインセグメント中をとびとびに検索し、文字間の白スペースを検出する。ここで、とびとびの検索とは、ラインセグメントのすべての画素を検索するのではなく、図22に示されるように、ラインセグメントの1つの段中の一部の画素271のみを検索するものである。なお、ラインセグメントの段中の全画素の1/3のみ、すなわち3画素ごとの検索で十分であるとの結果が得られている。画素271中で空白でない画素すなわち黒画素が検出されなかった場合には、272で示すように数列の画素をとびとびとして、例えば3画素ごとに新たな列で黒画素を検索する。ここで、画素を3列とびとびとして、非接触で重なっていない文字の検出性能は低下しないとの結果が得られている。このようなとびとびの検索を、図22の画素274のような最初の黒画素が検出されるまで繰り返す。

【0120】最初の黒画素が検出されるとステップS2102に処理を進め、ラインセグメントを後向きに検索し、完全に空白な列を検出する。ここでの検索は、ステップS2101の検索とは異なり、各列ごとにすべての画素を検索して完全に空白な列を検出するものである。すなわち、図22に示されるように、完全に空白な列276が検出されるまで後向きステップ275が実行される。

【0121】完全に空白な列が検出されると、処理をステップS2103に進め、完全に空白な列が検出されるまで画素274の列から前向きに検索される。ステップS2102と同様に、ここでの前向き検索も各列ごとにすべての画素を検索するもので、277で示すように前向きに検索される。図22の278で示されるような完全に空白な列が検出されるまで前向き検索が行われる。

【0122】ステップS2103で完全に空白な列が検出されると、処理をステップS2104に進め、空白列276と278とで文字を切り出す。その後、処理をステップS2101に戻し、空白でないすなわち黒画素が再び検出されるまで、ラインセグメントでとびとびの検索を再び行う。ラインセグメント全体で階層1の処理を終えると、テキストが単一スペース（クーリエフォントなど）であるか、あるいはテキストのスペースが未知もしくは単一スペース以外（比例フォントなど）であるかに応じて、図20の261あるいは264の階層2の処理に進める。テキストが単一スペースであれば、261の階層2の切り出しに処理を進める。

【0123】図23は単一スペーステキストのための階

層2の処理の流れを示すフローチャートである。なお、図23に示される処理ステップは、ROM17に保持されるプログラムステップに基づいてCPU10で実行される。階層2の処理に先立って、文字セグメントの幅を調べて、過小サイズの文字セグメントを割り出す。ここで、文字セグメントの幅が平均文字幅の半分以下のときに過小サイズであると判断する。過小サイズの文字セグメントが隣り合って存在する場合には、階層1の処理で1つの文字を半分ずつ2つに切り出してしまった可能性が高いため、この過小サイズ文字のペアを連結する。

【0124】ステップS2301では、各文字ブロックの幅をすべての文字ブロックの平均幅と比較して、階層1で切り出された文字ブロックのうち過大サイズのものを割り出す。ここで、各文字ブロックの幅とすべての文字ブロックの平均幅との比較は、文字が単一スペースであって各文字がほぼ同一の幅を有することが既知であるため、過大サイズの文字ブロックの検出には有効な処理となる。文字ブロックの幅（「W」）が以下の式を満たすと、文字ブロックが過大サイズであると判断される。

【0125】

$$[数5] W > (1+c) * W_{av}$$

ここで、cは定数、 W_{av} はラインセグメント中のすべての文字ブロックの平均幅である。なお、この判別処理はスケールに不変であることに注意されたい。定数cは、以下のように単一スペースフォントの統計的性質に基づいて決定される。クーリエアルファベットなどの単一スペースアルファベットの各文字は、単一スペースを有しており、各スペースは文字が存在する部分 α_i と文字の周りの空白スペースの部分 β_i とからなる。

【0126】例えば、図24の文字「e」に示されるように、文字「e」が存在するスペースは中心領域 α_i と周りの空白スペース β_i とからなる。ここで、「e」は「e」に対応する番号であり、すべてのiに対して $\alpha_i + \beta_i = 1$ である。 α_i と β_i についてはアルファベット中の各文字、すなわち英字、数字、記号などごとに求めることができ、 α_i と β_i の平均値（それぞれ α と β ）ならびに標準偏差（それぞれ σ_α と σ_β ）を計算できる。そこで、定数cは以下の式で求める。

【0127】

$$[数6] c = \sigma_\alpha / \alpha$$

図1の装置で用いるクーリエ文字セットでは、 $\alpha = 25/35$ 、 $\sigma_\alpha = 10/35$ となるため $c = 0.4$ となる。ステップS2301で過大サイズの文字ブロックが割り出されると、ステップS2302に処理を進め、過大サイズブロック中に含まれるおおよその文字数を算出し、おおよその文字境界を求める。具体的には、図24において、ブロック280の幅Wは、すべてのブロック280から283の平均の幅に $(1+c)$ を乗算して計算されるしきい値より大であるため、ブロック280は過大サイズの文字ブロックと判断される。そして、幅W

を α で割った値を最も近い整数値に丸めて、過大サイズブロック280中のおおよその文字数を算出する。

【0128】

【数7】文字数 $N = [W/\alpha]$ (最も近い整数)

また、ここで得られたブロック中のおおよその文字数「N」に基づいて、過大サイズのブロックを単一に分割して、おおよその文字境界を求める。ステップS2303では、ブロック中の画素の垂直投影特性284を、おおよその文字境界の近傍285で求める。ここで、垂直投影特性284を求める近傍は、距離 σ に基づいて決

められる。すなわち、図24に示されるように、おおよその文字境界の近傍 $\pm\sigma$ で垂直投影特性284を求める。
【0129】ステップS2304では、各垂直投影特性284中での最小位置286を割り出し、この最小位置284で文字の切り出しを行う。図20の261の階層2の処理を終えると、文字認識263、そしてさらに図2のステップS209に処理を進める。ラインセグメント中の文字のスペースが未知あるいは単一スペースでない場合には、文字がラインセグメント中で単一のスペースを有しているとは限らない。そこで、図20の264の階層2の処理に進み、非接触であるが重なっている文字間の切り出しを行う。図25と図26とはこの処理を説明する図である。

【0130】ステップS2501において、階層1で切り出された各文字を分析して、文字ブロック中の各画像の輪郭のアウトラインを追跡する処理を行う。すなわち、図26Aに示されるように、文字「fy」を有する文字ブロックは非接触だが重なっている文字「f」と「y」を含み、これらは重なっているため階層1の処理では切り出されない。そこで、図26Bに示されるように、まず、この文字ブロックをブロックの右下から左方向ならびに上方向に調べて黒画素を検出する。黒画素を検出すると、図26Cの287のように黒画素と黒画素とを接続したものである輪郭の追跡が行われる。第1の文字に対して全体の輪郭追跡を終えると、288のように文字ブロック中のすべての黒画素の輪郭を追跡するまで、スキャン処理を続ける。このような処理により、各々が分離した非接触の文字が得られ、図26Dに示されるようにこれらの文字が文字ブロックから切り出される。

【0131】階層2の処理では、非接触だが重なっている文字の切り出しとともに、複数のストロークからなる単一文字、例えば「i」、「j」、「:」、「;」、「!」、「=」、「%」をも分離してしまう。そこで、ステップS2502でこのような文字の再接続を行う。図27はこの処理の詳細なフローチャートである。図27に示す再接続処理の対象となるのは階層2の処理で切り出された文字のみであり、特定の条件を満たすときのみ再接続が行われる。具体的には、ブロックが重なる、

すなわち左側文字の最も右側の画素が右側文字の最も左側の画素の上あるいは下に位置するような場合にのみ、ブロックの再接続が行われる。

【0132】そこで、ステップS2701でブロックが重なっているかどうかを調べる。ブロックが重複していなければ、再接続処理は不必要であり(ステップS2702)、再接続処理を終了する。一方、ブロックが重複していれば、ステップS2703に処理を進め、ブロックが垂直方向に分離しているかどうかを調べる。ここで、ブロックが垂直方向に分離していると、「i」、「j」、「:」、「;」、「!」、「=」などの複数ストローク文字が階層2の処理で切り出されてしまった可能性があるため、これらの文字であるかどうかを調べる。

【0133】これらのブロックは垂直方向に分離されているため、高さがH1の上部ブロックと高さがH2の下部ブロックとを含む。これらの高さの計算はステップS2704で行われ、ステップS2705でH2が(2×H1)より大であれば文字「i」あるいは「j」が分離したものである可能性が高い。そこで、文字の重複度を計算する(ステップS2706)。すなわち、2つの部分に隣接する最も右側の4画素列を平均して、この平均位置の差を計算する。(全体位置ではなく最も右側の位置を平均したのは、文字「i」や「j」の上のドットが「i」や「j」の中心ではなく上部のセリフの右側に位置するためである)。この際、スケール不変の処理とするために、平均位置の差が本体のうちの小さい幅に定数を乗じたものより小であれば、ブロックを再接続するものとする。ここで、定数は、分離の程度が予測できないような劣化画像でも再接続できるように選択され、本実施例では「9/8」としている。

【0134】一方、ステップS2705でH2が(2×H1)以下、すなわち下部ブロックの高さが上部本体の2倍以下である場合には、「:」、「;」、「!」、「=」などの文字が切り出された可能性が高い。そこで、ステップS2707に処理を進め、この可能性を調べる。すなわち、各本体中の隣接する4画素列の平均中心値を求め、これらの中心値の差を得る。この際、スケール不変の処理とするために、中心値の差が2つの本体の幅のうちの小さい方に定数を乗じたものより小であれば、上述の文字の1つである可能性が高いためブロックを再接続するものとする。なお、上述のように、定数として「9/8」を用いると良好な結果が得られている。

【0135】ステップS2703でブロックが垂直方向に分離されない場合(すなわち、2つのブロック間に水平方向に伸びるギャップが存在しない場合)には、文字はタイプ1のパーセント記号(「タイプ1」)、タイプ2のパーセント記号(「タイプ2」)、タイプ3のパーセント記号(「タイプ3」)のどれかである可能性が高い。そこで、ステップS2708で以下のように順々に

各タイプごとにチェックする。なお、変数は以下の通りである。

【0136】W1:第1文字の幅(左から右)

W2:第2文字の幅

H1:第1文字の高さ(上から下)

H2:第2文字の高さ

L1:第1文字の左端画素の列

R1:第1文字の右端画素の列+1画素

L2:第2文字の左端画素の列

R2:第2文字の右端画素の列+1画素

注意: L1は常にL2以下である。

【0137】まず、タイプ1のパーセント記号をチェックする。以下の2つの条件が満たされるときにタイプ1のパーセント記号であると判断され、ブロックが連結される。

1) $0.24 < \min(W1, W2) / \max(W1, W2) < 0.77$

これはドット幅とライン幅との比の条件である。

【0138】2) $[\min(R1, R2) - \max(L1, L2)] / \min(W1, W2) > 0.76$

これはブロックが水平方向に大部分重なっているという条件である。次いで、タイプ2のパーセント記号をチェックする。以下の4つの条件が満たされるときにタイプ2のパーセント記号であると判断され、ブロックが連結される。

【0139】1) $(0.25) L2 < R1 - L2$

これはブロックが水平方向に十分重なっているという条件である。

2) $0.50 < W1/W2 < 1.10$

これはドット幅とライン幅の適切な比率の条件である。

3) $0.43 < (H1/H2) < 0.70$

これはドット高とライン高の適切な比率の条件である。

【0140】4) $(1/m) > 0.37$

ここで、mはパーセント記号の「斜線」部上のP1とP2とを結ぶ線の傾きである。なお、P1とP2とは以下の手法で求められる。

P1: P1は第2文字の上部からD行目の行で、第2文字のプリントテキストを含む左端の画素の位置である。

ここで、変数DはD = (0, 1) W2である。

【0141】P2: P2は第2文字の下部からD行目の行で、第2文字のプリントテキストを含む左端の画素の位置である。

さらに、タイプ3のパーセント記号をチェックする。以下の条件が満たされるときにタイプ3のパーセント記号であると判断され、ブロックが連結される。

1) $(0.25) L1 < R2 - L1$

これはブロックが水平方向に十分重なっているという条件である。

【0142】2) $0.50 < W2/W1 < 1.$

10

これはドット幅とライン幅の適切な比率の条件である。

3) $0.43 < (H2/H1) < 0.70$

これはドット高とライン高の適切な比率の条件である。

4) $(1/m) > 0.37$

ここで、mはパーセント記号の「斜線」部位上のP1とP2とを結ぶ線の傾きである。なお、P1とP2とは以下の手法で求められる。

【0143】P1: P1は第1文字の上部からD行目の行で、第2文字のプリントテキストを含む右端の画素の位置である。ここで、変数DはD = (0, 1) W2である。

P2: P2は第1文字の下部からD行目の行で、第2文字のプリントテキストを含む右端の画素の位置である。

図20の264で説明した(また、図23から図27で詳細に説明した)階層2の切り出し処理と再接続処理を終えると、265の認識処理を切り出された文字に対して行う。階層1と階層2の切り出しにおいてラインセグメント中のほとんどの文字は適切に切り出されているため、265の認識処理では階層1と階層2で切り出された文字のほとんどを認識することができる。これに対し、265で認識不能であった文字は、その文字ブロックに接触文字が含まれている可能性が高い。そこで、このような認識不能の文字ブロックに対して、266の階層3の切り出し処理を行い、接触文字を切り出す。

【0144】図28は階層3の切り出し処理を示すフローチャートであり、図29Aから図29Dは接触文字を階層3で切り出す処理を説明する図である。図28に示される処理ステップはROM17に保持されており、CPU10で実行される。一般に階層3の切り出し処理は、文字ブロックを斜めに切り出して行われる。斜めの切り出し線の傾きと位置とは、ブロック中の画素密度の垂直投影特性を求め、垂直投影特性中で最も深い谷の側面の傾きを求めることで算出される。そこで、再び画素密度の垂直方向以外の投影を行う。すなわち、垂直投影特性中の谷側面の傾きに対応する回転角度方向に画素密度の投影を行う。こうして得られた複数の密度投影の中での最小点を検出し、最小点を得られた角度と位置で切り出しを行う。以下、この処理について詳細な説明を行う。

【0145】ステップS2801において、画素密度の垂直投影特性を求める。例えば、図29Aに示されるように、接触文字「t1」に対して垂直投影特性を求める。ステップS2802では、垂直投影特性中の第1の谷を検出する。垂直投影特性はデジタルであるため(すなわち、離散的な画素数の和であるため)、滑らかではなく、谷は垂直投影特性中の最小値が第1の低い値より下であって、第2の高い値より大きい極大点で両側が囲まれた点であることで見付け出される。したがって、図29Aに示されるように、垂直投影特性を調べて、高い値292より高い上の点で囲まれてい

57

るような低いしきい値291より下の点が存在するかどうかを判別する。このような条件を満たす点が検出されれば、処理をステップS2803に進める。このような条件を満たす点が検出されなければ、以下に述べるようにしきい値の変更を行う。

【0146】まず、低いしきい値291を垂直投影特性の最大値の10%とし、高いしきい値292を垂直投影特性の最大値の20%とする。ここで、高いしきい値と低いしきい値に関する条件を満たす点が検出されない場合には、高いしきい値と低いしきい値との双方とも垂直投影特性の最大値の2%だけ増加させる。図29Aでは、しきい値291と292の条件を満たす点は検出されない。そこで、しきい値を図29Bに示すように増加させて、低いしきい値291以下であって、高いしきい値292より高い点295と296が両側に存在するような点294を検出する。点294を検出すると、ステップS2803に処理を進め、点294を囲む谷側面の傾きを算出する。谷の右側面の傾きは点294と295とを結ぶ線の傾きであり θ_1 で示される。同様に、谷の左側面の傾きは点294と296とを結ぶ線の傾きであり θ_2 で示される。

【0147】そこで、ステップS2804に処理を進め、角度 θ_1 と θ_2 および角度 θ_1 と θ_2 の近傍で回転させて投影特性を求める。すなわち、回転投影特性を角度 θ_1 、 $\theta_1 \pm 3^\circ$ 、 $\theta_1 \pm 6^\circ$ 、 θ_2 、 $\theta_2 \pm 3^\circ$ 、 $\theta_2 \pm 6^\circ$ で求める。この回転投影特性は文字ブロック中の画素を三角関数変換することで求められる。より簡易な手法として、各回転角度（最も近い角度に近似）ごとに画素位置が示される表を用意して、テーブルルックアップで回転投影特性を求めることもできる。なお、回転投影特性の各点は、この画素位置の和として求められる。

【0148】図29Cの297は回転投影特性の典型例を示したものである。回転投影特性297上の各点は、ここでは θ_1 の回転方向の画素数の和として求められる。上述のように、ここでの和は、文字ブロック中の画像を三角関数変換することで求めることもできるし、各回転角度ごとに用意された表を参照して簡易に求めることもできる。

【0149】図29CとDの点線で示されるようなすべての回転投影特性を求めると、ステップS2805に処理を進め、各回転投影特性（10個すべて）ならびにステップS2801で求めた垂直投影特性とを比較して、すべての投影特性の中で最小点を検出する。最小点となる投影特性の角度が切り出し角度に対応する。すなわち、図29Cに示されるように、点299が11個の投影特性の中で最小点となれば、文字ブロックの切り出しが角度 θ_1 で最小点299の位置で行われる（ステップS2806）。

【0150】階層3の切り出し処理を終えると、切り出

58

された文字に対して認識処理267を行う。ここで、階層3で切り出された2つの文字ブロックとも認識が行われたならば、図2のステップS209に処理を進める。しかし、ここでもなお認識不能なブロックが残る可能性があり、不適切な切り出しが行われた可能性と切り出された文字ブロックを再接続すべきであるという点を考慮する必要がある。この処理は再接続ブロック269で行われ、図30に詳細に示されている。

【0151】ステップS3001では、階層3での切り出し部分の双方に対して267の認識処理を試みる。ステップS3002で2つの要素の認識が可能であると判断されると、上述の図2のステップS209に処理を進める。一方、2つの切り出し要素とも認識不能である場合には、ステップS3003において少なくとも1つの要素が認識可能であるかどうかを判別する。2つの要素とも認識不能である場合にはステップS3004に進み、各要素に対して階層3の切り出し処理とステップS3001などの処理とを行う。

【0152】一方、ステップS3003において少なくとも1つの要素が認識可能であると判断されると、ステップS3005に処理を進め、認識不能の要素に対してさらに階層3の切り出し処理を行う。そして、ステップS3006で新たに切り出された要素が認識可能であると判断されれば、認識不能な要素はなくなり図2のステップS209に処理を進める。これに対して、新たに切り出された要素がともに認識不能であるとステップS3006で判断されると、ステップS3007に処理を進め、不適切な切り出しブロックを再接続することを考える。

【0153】図31は、階層3の処理において不適切な切り出しが行われる可能性を説明する図である。図31Aはイタリック文字「hm」がかなり劣化している例を示したものである。図20の266の階層3の処理では第1の切り出しを301で行うため、文字「h」の垂直部とループ部とが分離してしまう。そして、切り出された各要素に対して267で認識処理を行い、第1のブロックは「l」として認識され、第2のブロックは認識不能と判断されたとする。

【0154】このような場合、処理はステップS3005に進み、認識不能の要素303に対して図31Bのようにさらなる階層3の切り出し処理が行われる。階層3の処理により、304でさらなる切り出しが行われ、切り出された要素305と306に対して267の認識処理が行われる。しかし、要素305は文字として認識不能であるため、再接続処理が必要であると判断される。

【0155】そこで、ステップS3007において、認識不能の切り出し要素を、あらかじめ切り出された隣接する要素と接続する。この際、隣接する要素は認識されたものでも、認識不能のものでも良い。すると、図31Cのように、要素302を要素305と再接続して文字

「h」をほとんど含む新たな要素302'が生成される。そこで、要素302'と要素306とに対して267の認識処理を行う(ステップS3008)。

【0156】これらの処理を終えると、ステップS3001に処理を進め、2つの要素ともに認識されたかどうかを調べる。この例の場合には、2つの要素とも文字「h」、「m」として認識されたため処理を終了する。一方、2つの要素とも認識不能であった場合には、上述の処理を繰り返す。

【0157】

【発明の効果】以上説明したように、本発明により、高速かつ正確に文書上の文字を認識してテキストファイルを作成する文字認識方法及び装置を提供できる。また、高速かつ正確に文書上のテキストと非テキストとを選別して、テキストブロックを割り出す方法及び装置を提供できる。

【0158】また、高速かつ正確に文書上の非テキストを分類できる方法及び装置を提供できる。また、高速かつ正確にテキストブロックからテキストラインを分割できる方法及び装置を提供できる。また、高速かつ正確に傾いた文書のテキストブロックからテキストラインを分割できる方法及び装置を提供できる。

【0159】また、高速かつ正確にテキストラインから文字を切り出す方法及び装置を提供できる。また、不適切に切り出された文字の再切り出しが可能な方法及び装置を提供できる。

【図面の簡単な説明】

【図1】本実施例の装置のブロック図である。

【図2】文字認識処理の流れを示すフローチャートである。

【図3】本実施例に基づくブロック分類と選別の処理の流れを示すフローチャートである。

【図4A】画素画像データ中の連結部位の分類処理の流れを示すフローチャートである。

【図4B】画素画像データ中の連結部位の分類処理の流れを示すフローチャートである。

【図4C】画素画像データ中の連結部位の分類処理の流れを示すフローチャートである。

【図5A】輪郭追跡を説明するための図である。

【図5B】輪郭追跡を説明するための図である。

【図5C】輪郭追跡を説明するための図である。

【図6A】非テキスト部位の分類処理を説明するための図である。

【図6B】非テキスト部位の分類処理を説明するための図である。

【図6C】非テキスト部位の分類処理を説明するための図である。

【図7A】白輪郭処理を説明するための図である。

【図7B】白輪郭処理を説明するための図である。

【図7C】白輪郭処理を説明するための図である。

【図7D】白輪郭処理を説明するための図である。

【図8】他のテキスト部位のサイズと近さに基づいてテキスト部分を水平方向に選択的に連結してテキストラインを形成する処理の流れを示すフローチャートである。

【図9】他のテキストラインのサイズと近さに基づいてテキストラインを垂直方向に選択的に連結してテキストブロックを形成する処理の流れを示すフローチャートである。

【図10】代表的な画像の画素データを示す図である。

10 【図11】ブロック分類と選別処理を説明するための図である。

【図12】ブロック分類と選別処理を説明するための図である。

【図13】ブロック分類と選別処理を説明するための図である。

【図14】図11から13にそれぞれ対応する典型的な階層の木構造である。

【図15】図11から13にそれぞれ対応する典型的な階層の木構造である。

20 【図16】図11から13にそれぞれ対応する典型的な階層の木構造である。

【図17】本実施例のライン分割処理の流れを示すフローチャートである。

【図18A】本実施例のライン分割処理を説明するための図である。

【図18B】本実施例のライン分割処理を説明するための図である。

【図18C】本実施例のライン分割処理を説明するための図である。

30 【図18D】本実施例のライン分割処理を説明するための図である。

【図19A】本実施例のライン分割処理を説明するための図である。

【図19B】本実施例のライン分割処理を説明するための図である。

【図19C】本実施例のライン分割処理を説明するための図である。

【図19D】本実施例のライン分割処理を説明するための図である。

40 【図20】本実施例の文字切り出し処理の機能ブロック図である。

【図21】図20の階層1の文字切り出し処理の流れを示すフローチャートである。

【図22】階層1の切り出し処理を説明するための図である。

【図23】図20における単一スペースモード(クーリエフォントなど)の階層2の文字切り出し処理の流れを示すフローチャートである。

50 【図24】階層2の文字切り出し処理を説明するための図である。

【図25】図20における全スペースモード（比例スペースなど）の階層2の文字切り出し処理の流れを示すフローチャートである。

【図26A】階層2の文字切り出し処理を説明するための図である。

【図26B】階層2の文字切り出し処理を説明するための図である。

【図26C】階層2の文字切り出し処理を説明するための図である。

【図26D】階層2の文字切り出し処理を説明するための図である。 10

【図27】階層2の処理で切り出された複数ストローク文字を再融合するための再融合手法を示すフローチャートである。

【図28】図20の階層3の処理の流れを示すフローチャートである。

【図29A】階層3の文字切り出し処理を説明するための図である。

【図29B】階層3の文字切り出し処理を説明するための図である。

【図29C】階層3の文字切り出し処理を説明するための図である。

【図29D】階層3の文字切り出し処理を説明するための図である。

【図30】図20の階層3の処理で切り出された部位の再融合処理の流れを示すフローチャートである。

【図31A】再連結処理を説明するための図である。

【図31B】再連結処理を説明するための図である。

【図31C】再連結処理を説明するための図である。

【図32】文字認識すべき文書のページの代表例である。 30

【図33A】従来のライン分割手法を説明するための図である。

【図33B】従来のライン分割手法を説明するための図である。

【図33C】従来のライン分割手法を説明するための図である。

【図34A】従来の文字切り出し手法を説明するための図である。

【図34B】従来の文字切り出し手法を説明するための図である。 40

【図35】本実施例のブロック選別プログラムのソースコードを示す図である。

【図36】本実施例のブロック選別プログラムのソースコードを示す図である。

【図37】本実施例のブロック選別プログラムのソースコードを示す図である。

【図38】本実施例のブロック選別プログラムのソースコードを示す図である。

【図39】本実施例のブロック選別プログラムのソース 50

コードを示す図である。

【図40】本実施例のブロック選別プログラムのソースコードを示す図である。

【図41】本実施例のブロック選別プログラムのソースコードを示す図である。

【図42】本実施例のブロック選別プログラムのソースコードを示す図である。

【図43】本実施例のブロック選別プログラムのソースコードを示す図である。

【図44】本実施例のブロック選別プログラムのソースコードを示す図である。

【図45】本実施例のブロック選別プログラムのソースコードを示す図である。

【図46】本実施例のブロック選別プログラムのソースコードを示す図である。

【図47】本実施例のブロック選別プログラムのソースコードを示す図である。

【図48】本実施例のブロック選別プログラムのソースコードを示す図である。

20 【図49】本実施例のブロック選別プログラムのソースコードを示す図である。

【図50】本実施例のブロック選別プログラムのソースコードを示す図である。

【図51】本実施例のブロック選別プログラムのソースコードを示す図である。

【図52】本実施例のブロック選別プログラムのソースコードを示す図である。

【図53】本実施例のブロック選別プログラムのソースコードを示す図である。

【図54】本実施例のブロック選別プログラムのソースコードを示す図である。

【図55】本実施例のブロック選別プログラムのソースコードを示す図である。

【図56】本実施例のブロック選別プログラムのソースコードを示す図である。

【図57】本実施例のブロック選別プログラムのソースコードを示す図である。

【図58】本実施例のブロック選別プログラムのソースコードを示す図である。

【図59】本実施例のブロック選別プログラムのソースコードを示す図である。

【図60】本実施例のブロック選別プログラムのソースコードを示す図である。

【図61】本実施例のブロック選別プログラムのソースコードを示す図である。

【図62】本実施例のブロック選別プログラムのソースコードを示す図である。

【図63】本実施例のブロック選別プログラムのソースコードを示す図である。

【図64】本実施例のブロック選別プログラムのソース

コードを示す図である。

【図 90】本実施例のブロック選別プログラムのソースコードを示す図である。

【図 9 1】本実施例のブロック選別プログラムのソースコードを示す図である。

【図 9 2】本実施例のブロック選別プログラムのソースコードを示す図である。

【図 9 3】本実施例のブロック選別プログラムのソースコードを示す図である。

【図94】本実施例のブロック選別プログラムのソースコードを示す図である。

【図 9 5】本実施例のブロック選別プログラムのソースコードを示す図である。

【図 96】本実施例のブロック選別プログラムのソースコードを示す図である。

【図 97】本実施例のブロック選別プログラムのソースコードを示す図である。

【図 98】本実施例のブロック選別プログラムのソースコードを示す図である。

【図 9 9】本実施例のブロック選別プログラムのソースコードを示す図である。

【図100】本実施例のブロック選別プログラムのソースコードを示す図である。

【図 101】本実施例のブロック選別プログラムのソースコードを示す図である。

【図102】本実施例のブロック選別プログラムのソースコードを示す図である。

【図103】本実施例のブロック選別プログラムのソースコードを示す図である。

【図104】本実施例のブロック選別プログラムのソースコードを示す図である。

【図105】本実施例のブロック選別プログラムのソースコードを示す図である。

【図106】本実施例のブロック選別プログラムのソースコードを示す図である。

【図107】本実施例のブロック選別プログラムのソースコードを示す図である。

【図108】本実施例のブロック選別プログラムのソースコードを示す図である。

【図109】本実施例のブロック選別プログラムのソースコードを示す図である。

【図 110】本実施例のブロック選別プログラムのソースコードを示す図である。

【図 111】本実施例のブロック選別プログラムのソースコードを示す図である。

【図 1 1 2】本実施例のブロック選別プログラムのソースコードを示す図である。

【図 1-13】本実施例のブロック選別プログラムのソースコードを示す図である。

【図 114】本実施例のブロック選別プログラムのソー

66

スコードを示す図である。

【図140】本実施例の比例スペースプログラムのソースコードを示す図である。

【図141】本実施例の比例スペースプログラムのソースコードを示す図である。

【図 142】本実施例の比例スペースプログラムのソースコードを示す図である。

【図143】本実施例の比例スペースプログラムのソースコードを示す図である。

【図 1.4 4】本実施例の比例スペースプログラムのソースコードを示す図である。

【図145】本実施例の比例スペースプログラムのソースコードを示す図である。

【図146】本実施例の比例スペースプログラムのソースコードを示す図である。

【図14.7】本実施例の比例スペースプログラムのソースコードを示す図である。

【図148】本実施例の比例スペースプログラムのソースコードを示す図である。

【図14.9】本実施例の比例スペースプログラムのソースコードを示す図である。

【図150】本実施例の比例スペースプログラムのソースコードを示す図である。

【図151】本実施例の比例スペースプログラムのソースコードを示す図である。

【図 1.5 2】本実施例の比例スペースプログラムのソースコードを示す図である。

【図153】本実施例の比例スペースプログラムのソースコードを示す図である。

【図154】本実施例の比例スペースプログラムのソースコードを示す図である。

【図155】本実施例の比例スペースプログラムのソースコードを示す図である。

【図156】本実施例の比例スペースプログラムのソースコードを示す図である。

【図157】本実施例の比例スペースプログラムのソースコードを示す図である。

【図158】. 本実施例の比例スペースプログラムのソースコードを示す図である。

【図159】本実施例の比例スペースプログラムのソースコードを示す図である。

【図160】本実施例の比例スペースプログラムのソースコードを示す図である。

【図161】本実施例の比例スペースプログラムのソースコードを示す図である。

【図162】本実施例の比例スペースプログラムのソースコードを示す図である。

【図163】本実施例の比例スペースプログラムのソースコードを示す図である。

【図164】本実施例の比例スペースプログラムのソー

70

グラムのソースコードを示す図である。

【図 2 3 2】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図 2 3 3】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図 234】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図 235】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図 236】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図 2 3 7】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図 238】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図 239】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図240】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図 241】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図 2 4 2】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図243】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

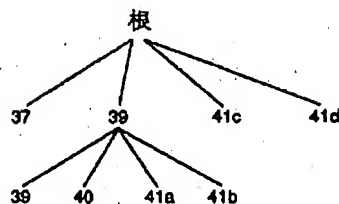
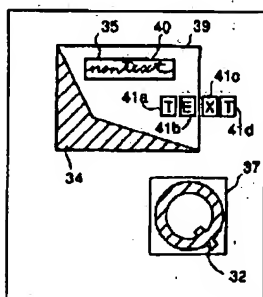
【図 2 4-4】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図 245】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図 2 4 6】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図2-47】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【☒ 26 B】



fy

【☒ 26 C】

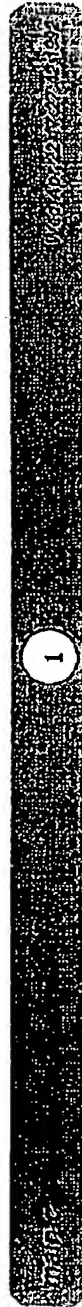
fy

【例 126】

```
#define FILE_LENGTH
#define LINE_LENGTH
#define FONT_SIZE_LENGTH
```

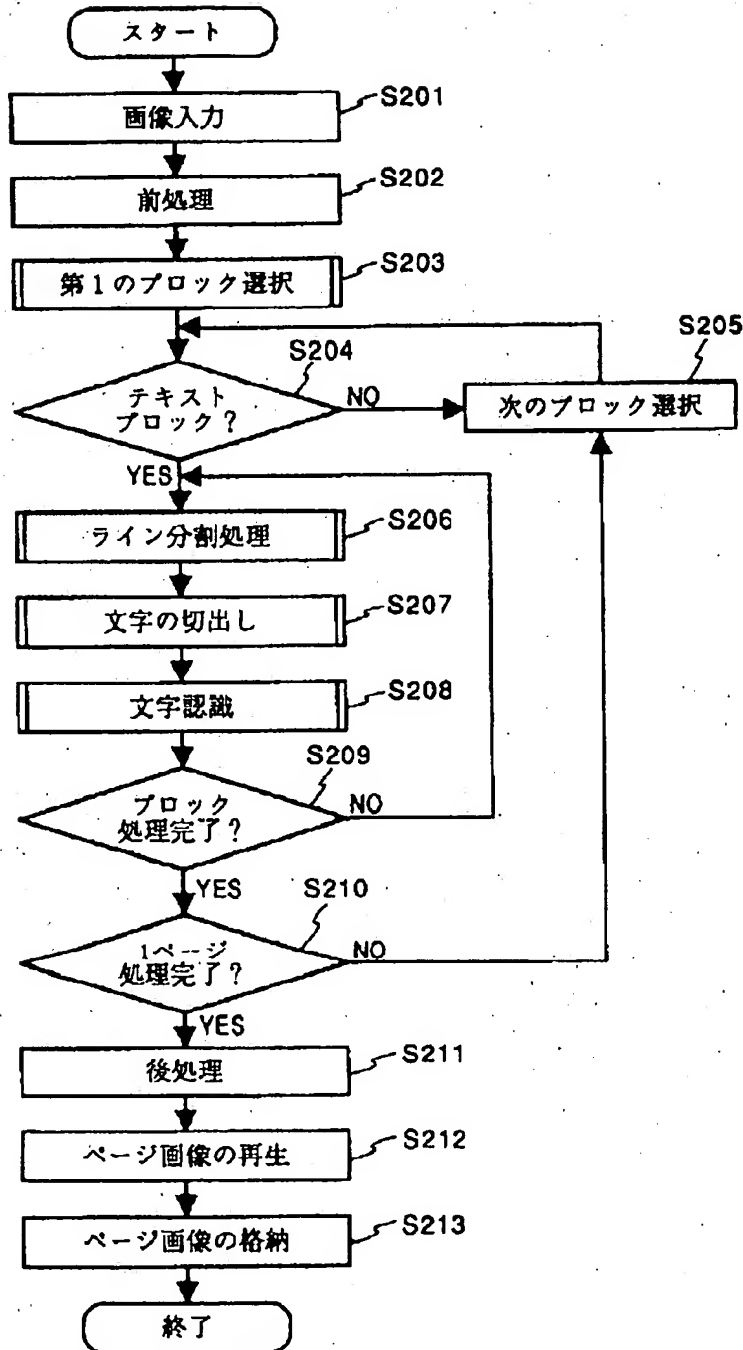
20
21
20

【~~3~~ 3 6】

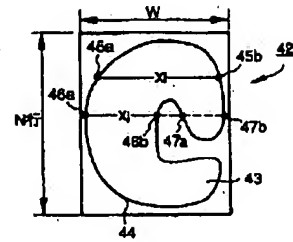


```
#include "util.h"
main(argc,argv)
unsigned int argc;
char *argv;
{
    struct sv, pobj_blocked();
    /*
     * Initialize interface
     */
    UtopLevel = Uninitialise('OCR Block Selection Demo', argc, argv);
    /*
     * Create and popen pipes for the interface
     */
    sv = pobj_blocked();
    /*
     * Enter the event loop, this function never returns.
     */
    while(loop{});
}
```

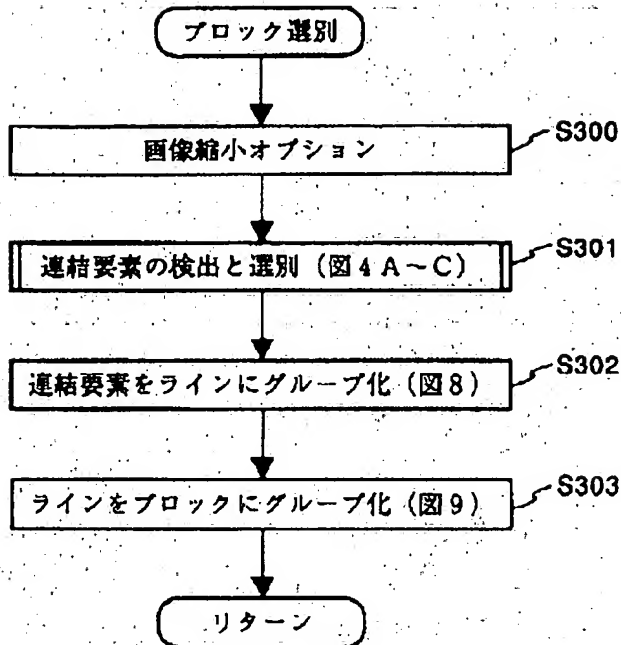
【図2】



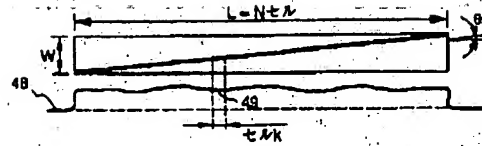
【図6A】



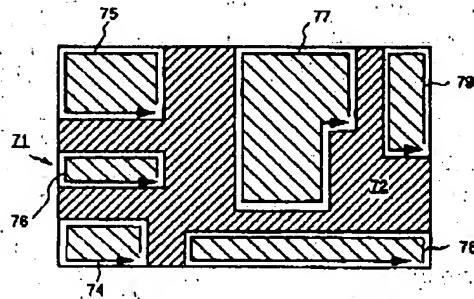
【図3】



【図6 B】



【図7 B】



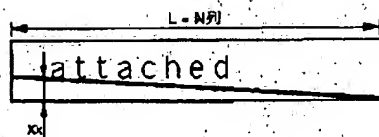
【図194】

```

free(char ** p);

```

【図6 C】



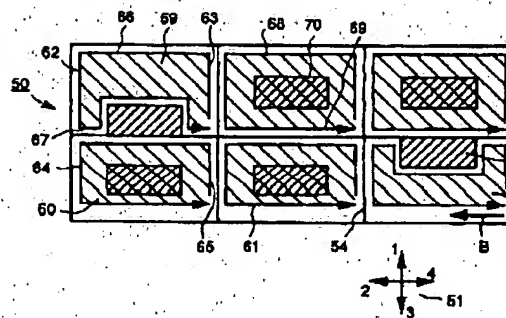
【図234】

```

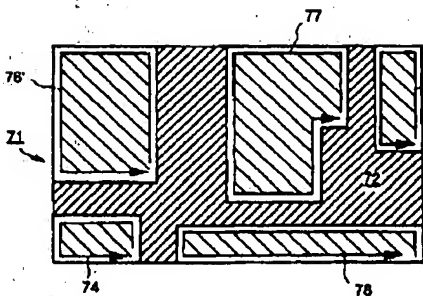
free(char ** p);

```

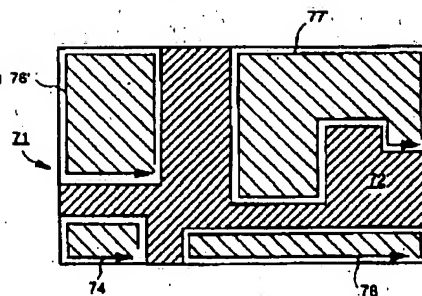
【図7 A】



【図7 C】



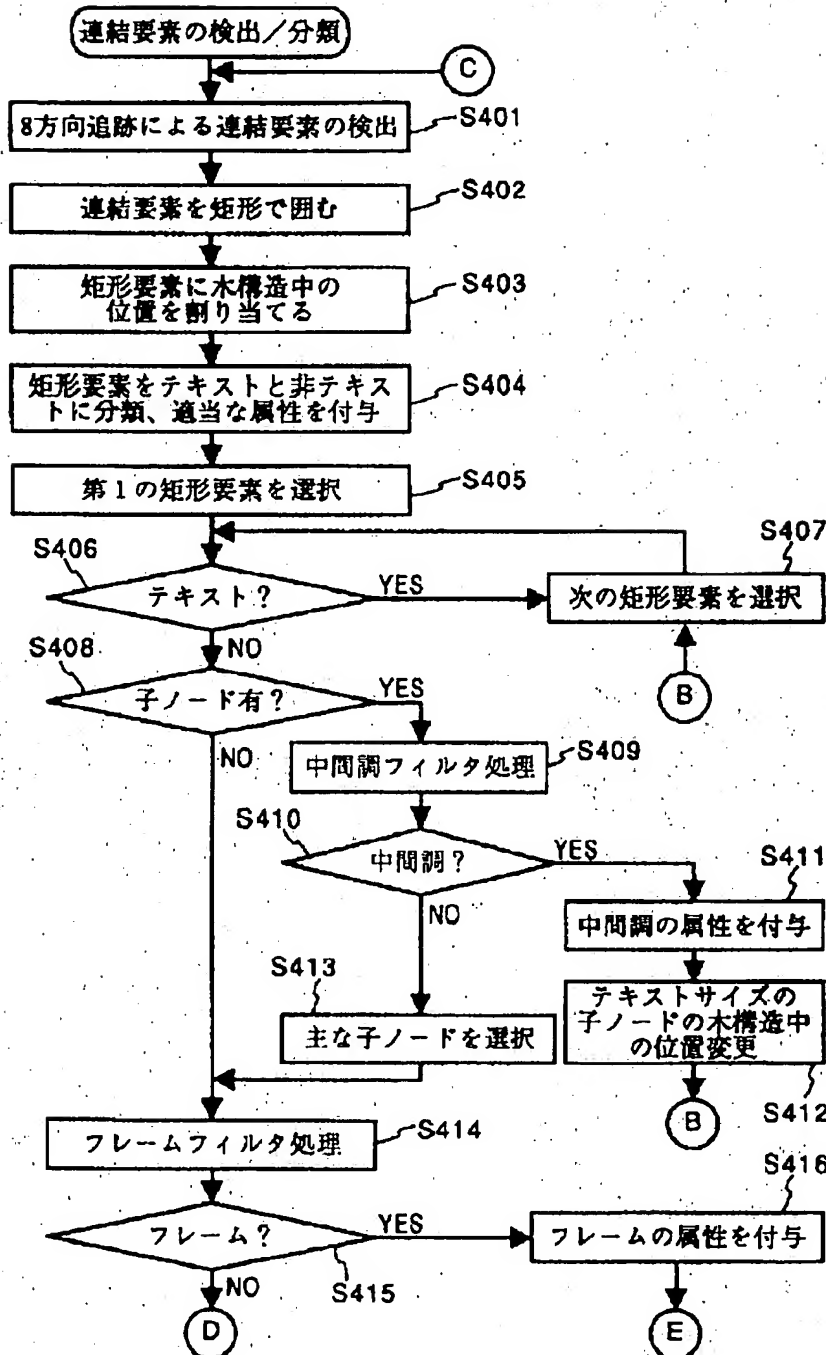
【図7 D】



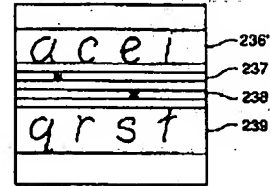
【図26 D】



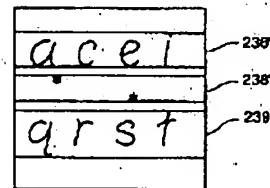
【図4A】



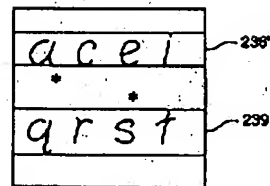
【図18B】



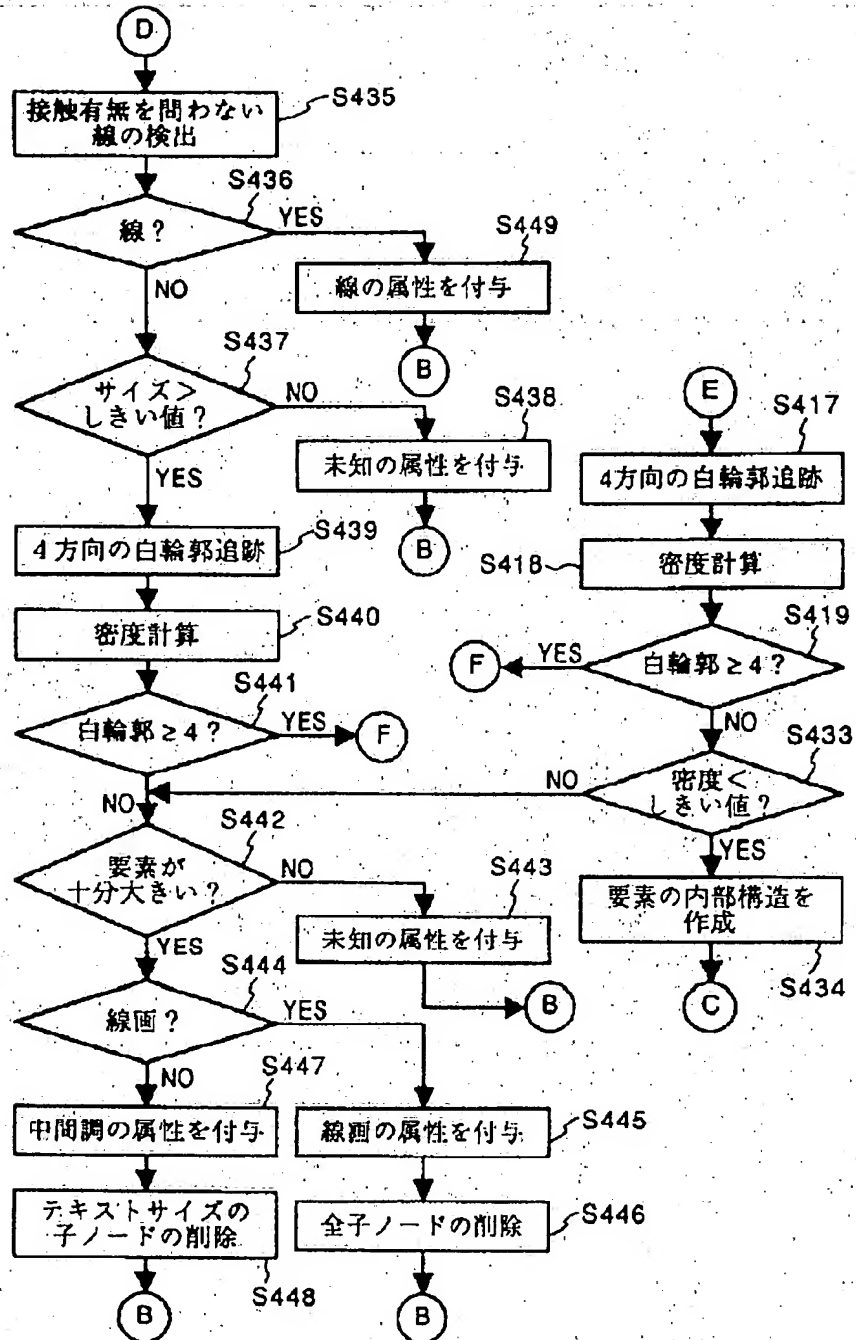
【図18C】



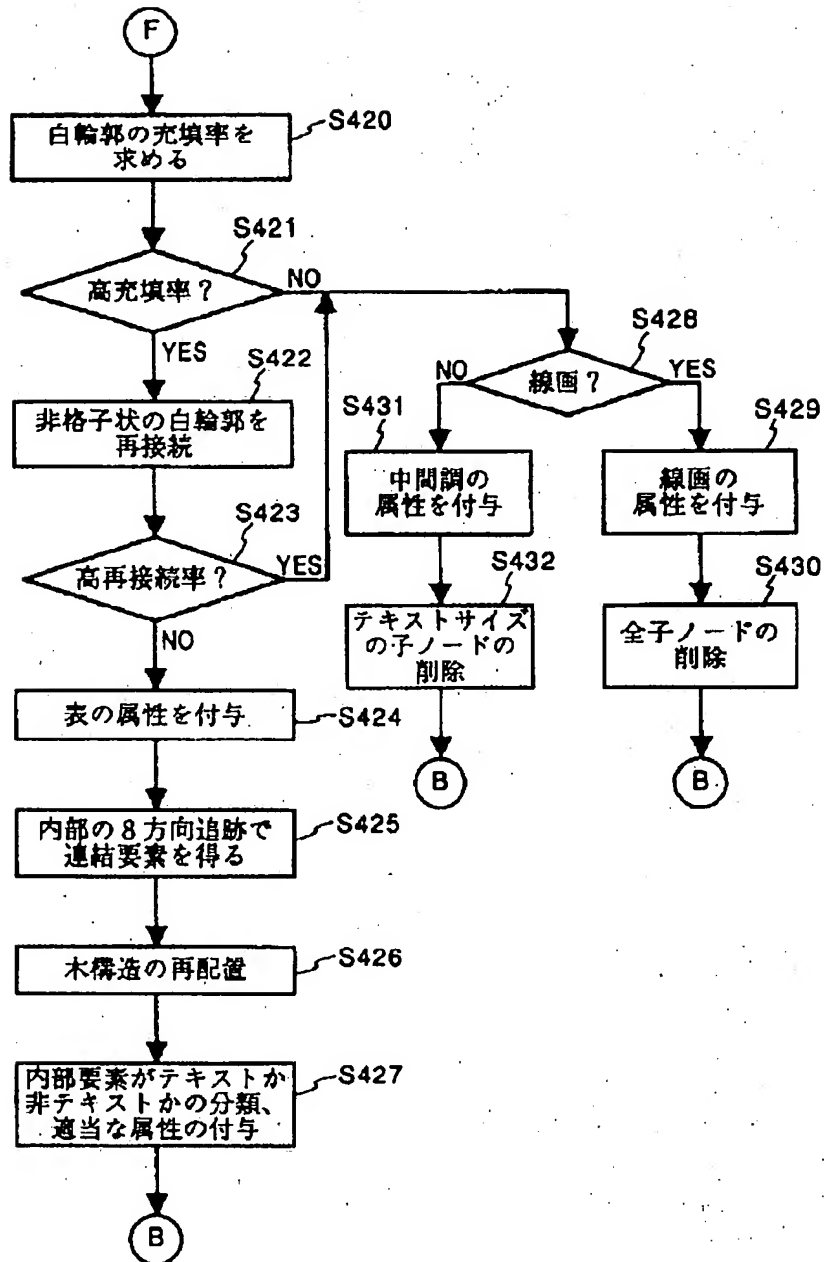
【図18D】



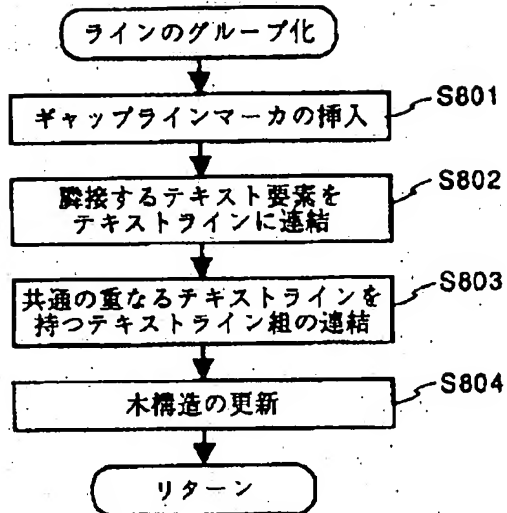
【図4B】



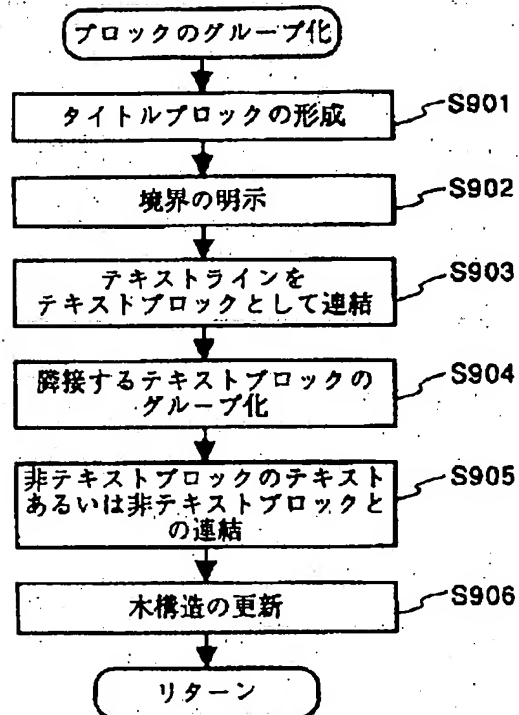
【図4C】



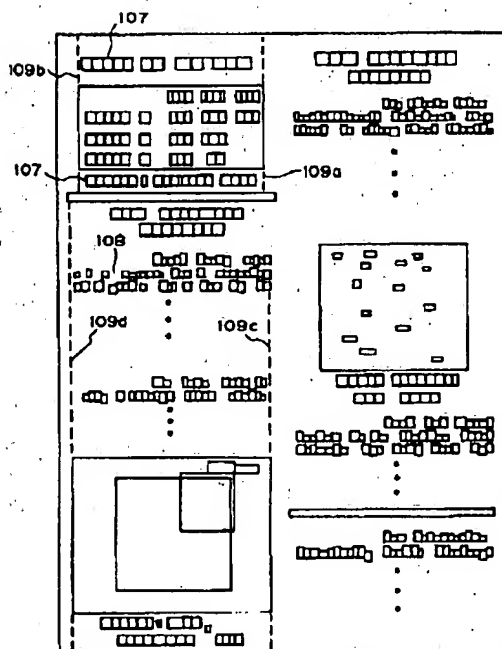
【図8】



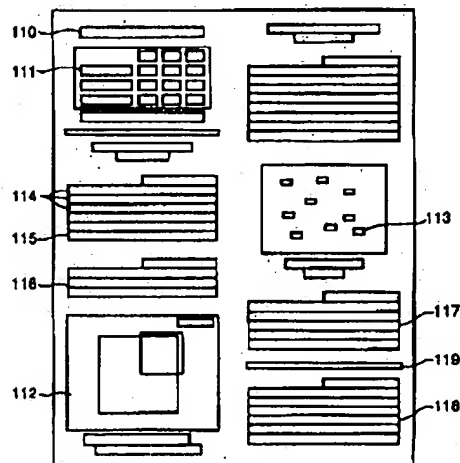
【図9】



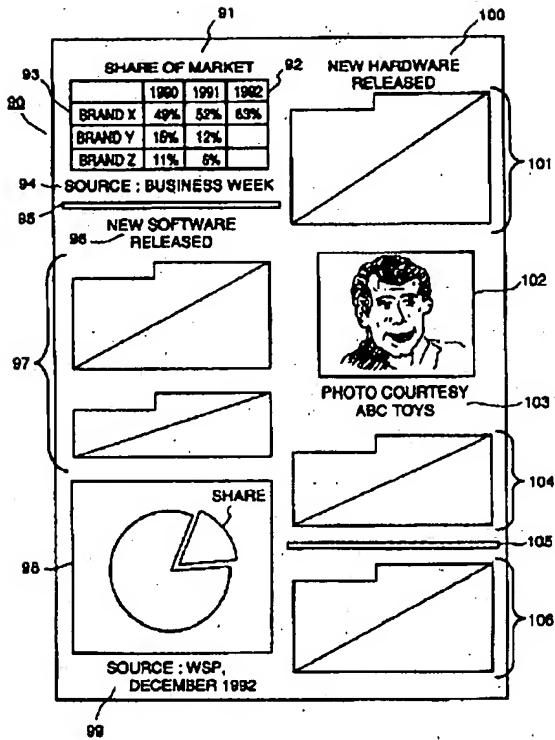
【図11】



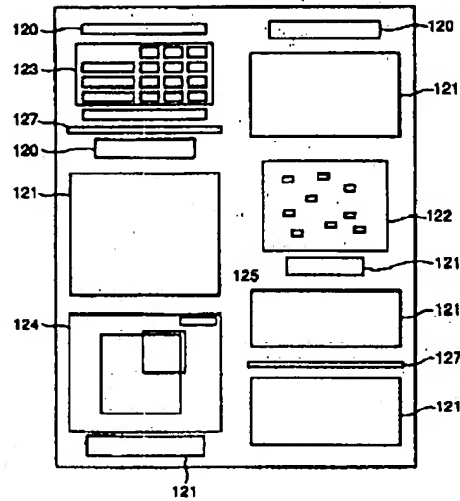
【図12】



【図10】



【図13】



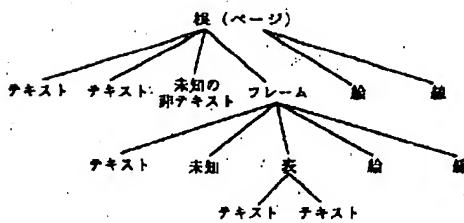
【図26A】

Satisfy

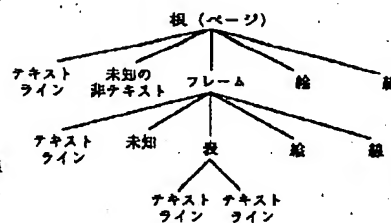
【図35】

Source Code For
Block Selection

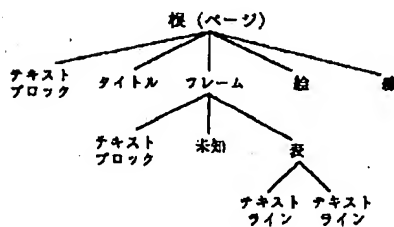
【図14】



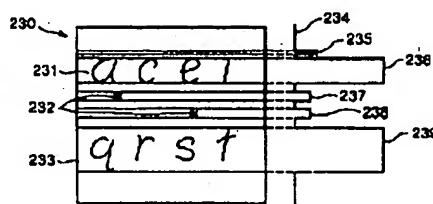
【図15】



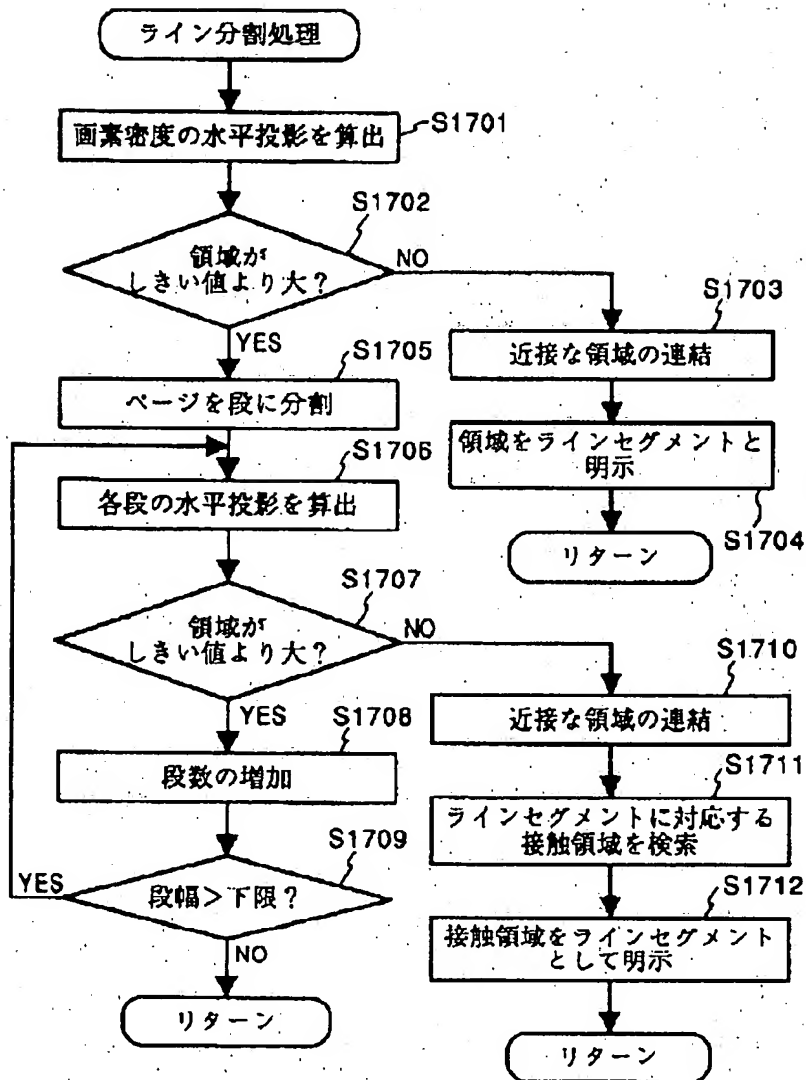
【図16】



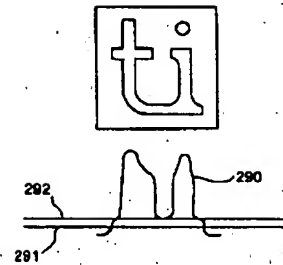
【図18A】



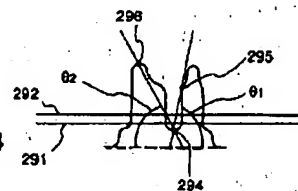
【図17】



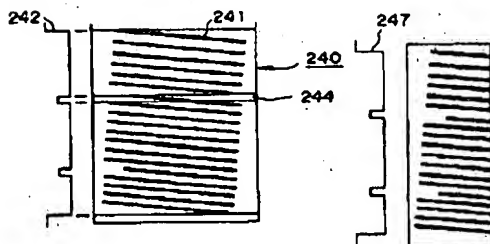
【図29A】



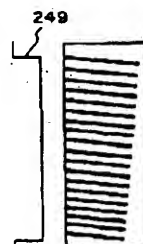
【図29B】



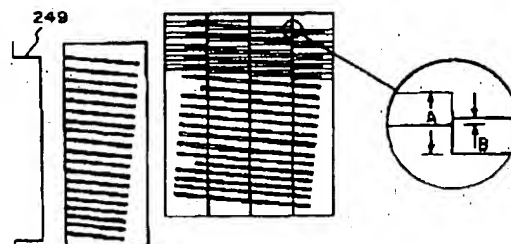
【図19A】



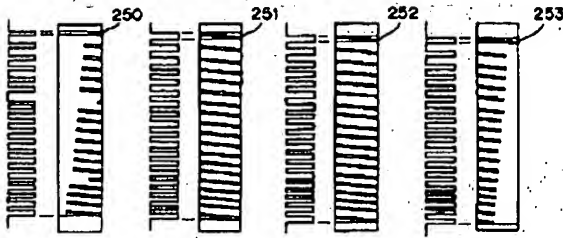
【図19B】



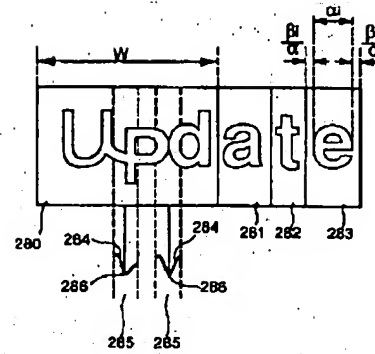
【図19D】



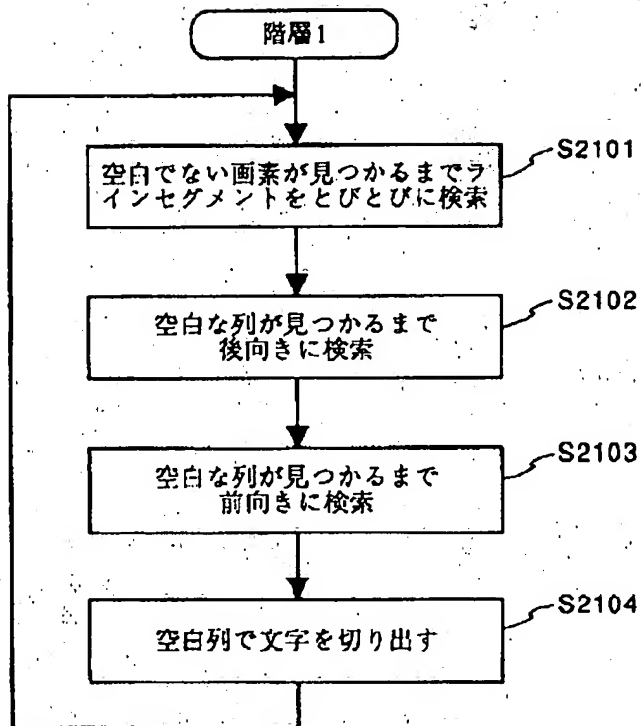
【図19C】



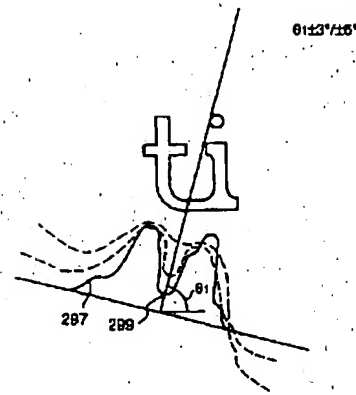
【図24】



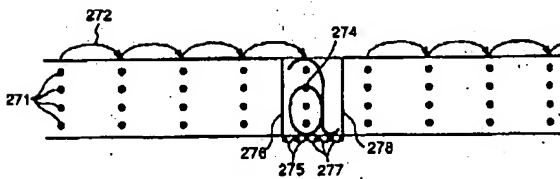
【図21】



【図29C】



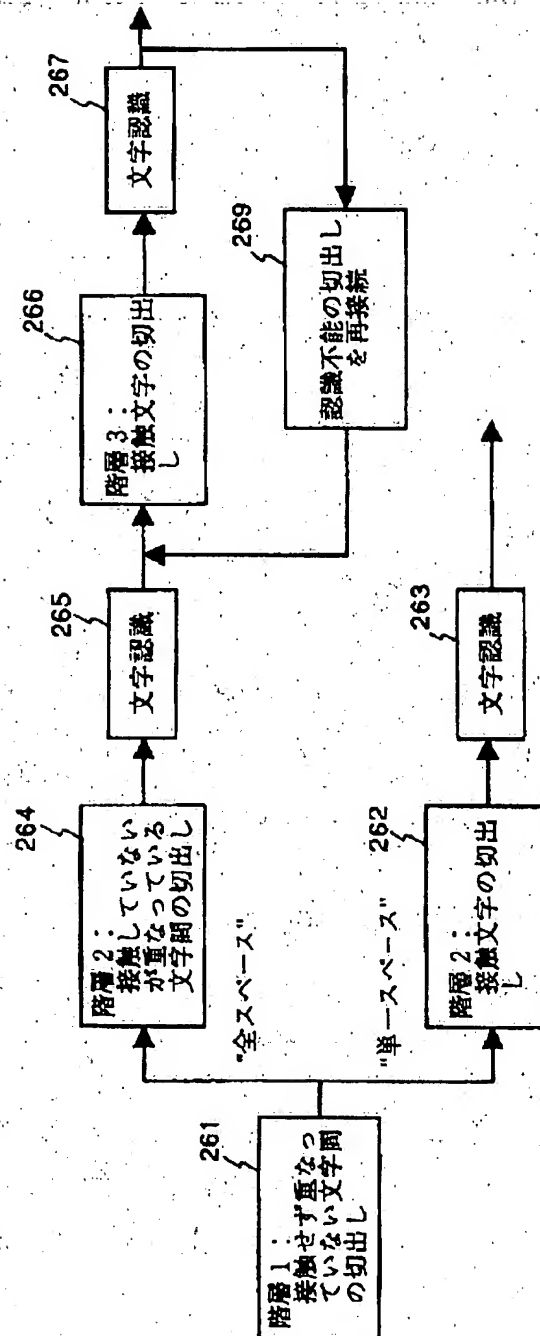
【図22】



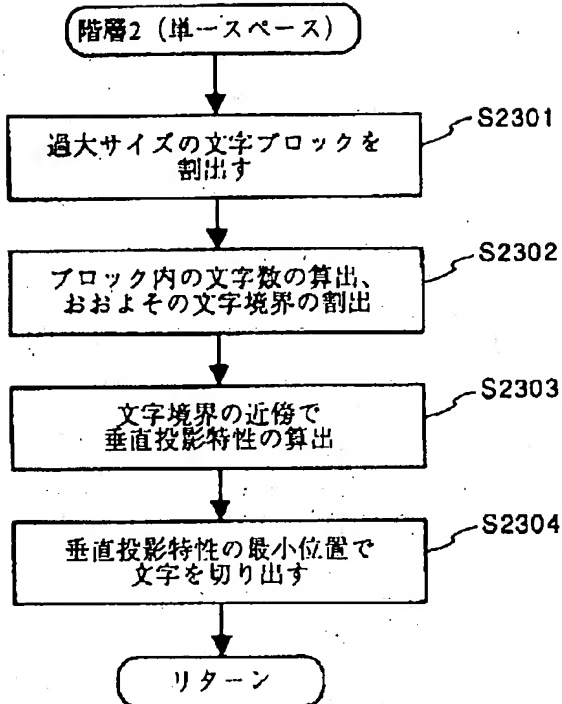
【図29D】



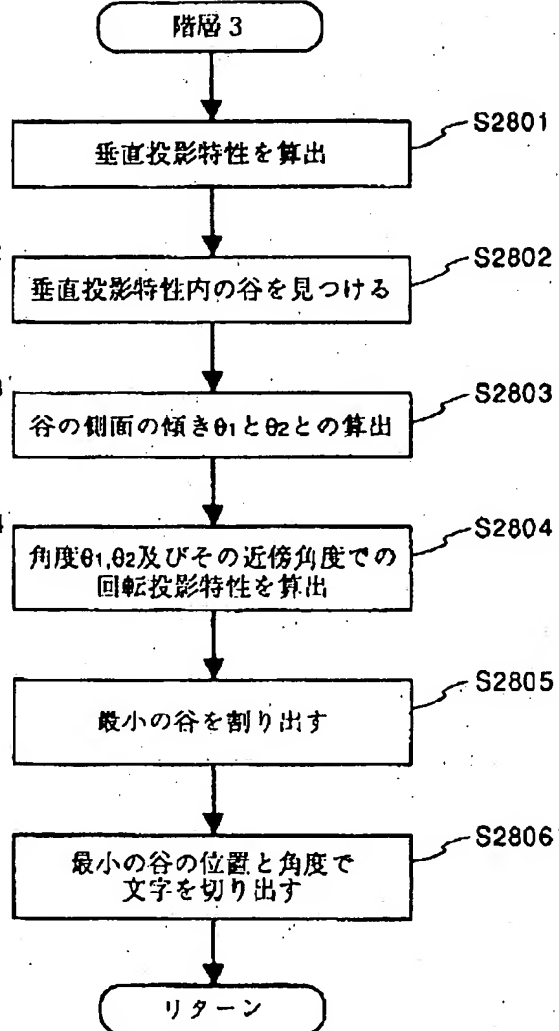
【図20】



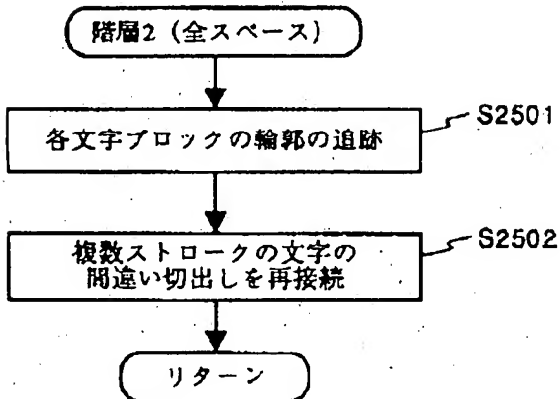
【図23】



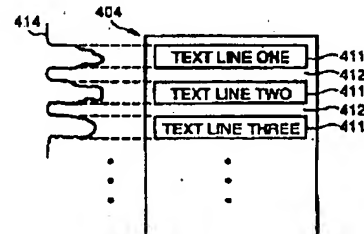
【図28】



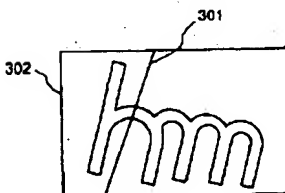
【図25】



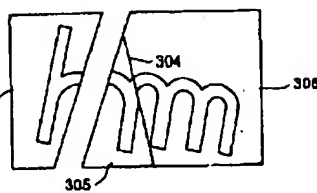
【図33A】



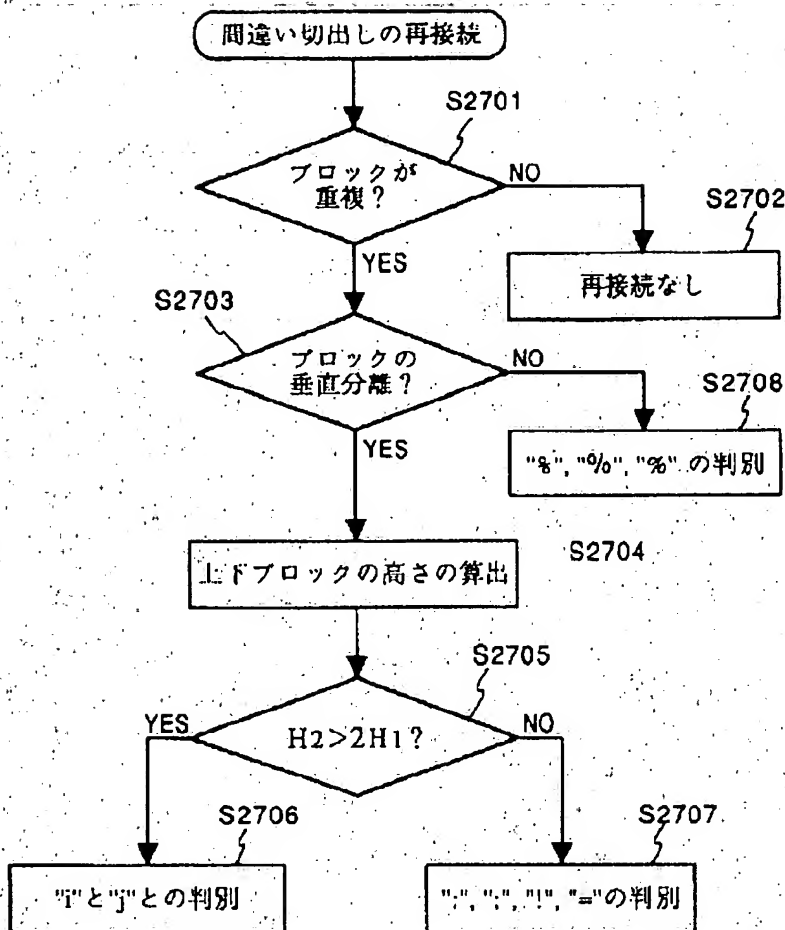
【図31A】



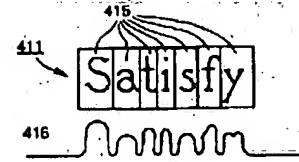
【図31B】



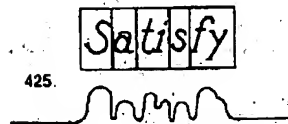
【図27】



【図34A】



【図34B】



【図134】

```

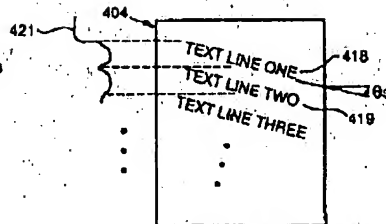
extern float *vector();
extern unsigned char *cvector();
extern int *ivector();
extern double *dvector();
extern unsigned char **cmatrix();
extern float **matrix();
extern int **imatrix();
extern double **dmatrix();
extern void free_vector();
extern void free_cvector();
extern void free_ivector();
extern void free_dvector();
extern void free_matrix();
extern void free_imatrix();
extern void free_dmatrix();
extern void free_cmatrix();

```

【図31C】



【図33B】



【図129】

```

struct sectionblock *previous;
struct sectionblock *next;

int upper_y, lower_y, left_x, right_x;
int width, length;
int index;
int multilineblock;
int size;
int boundary_index;

```

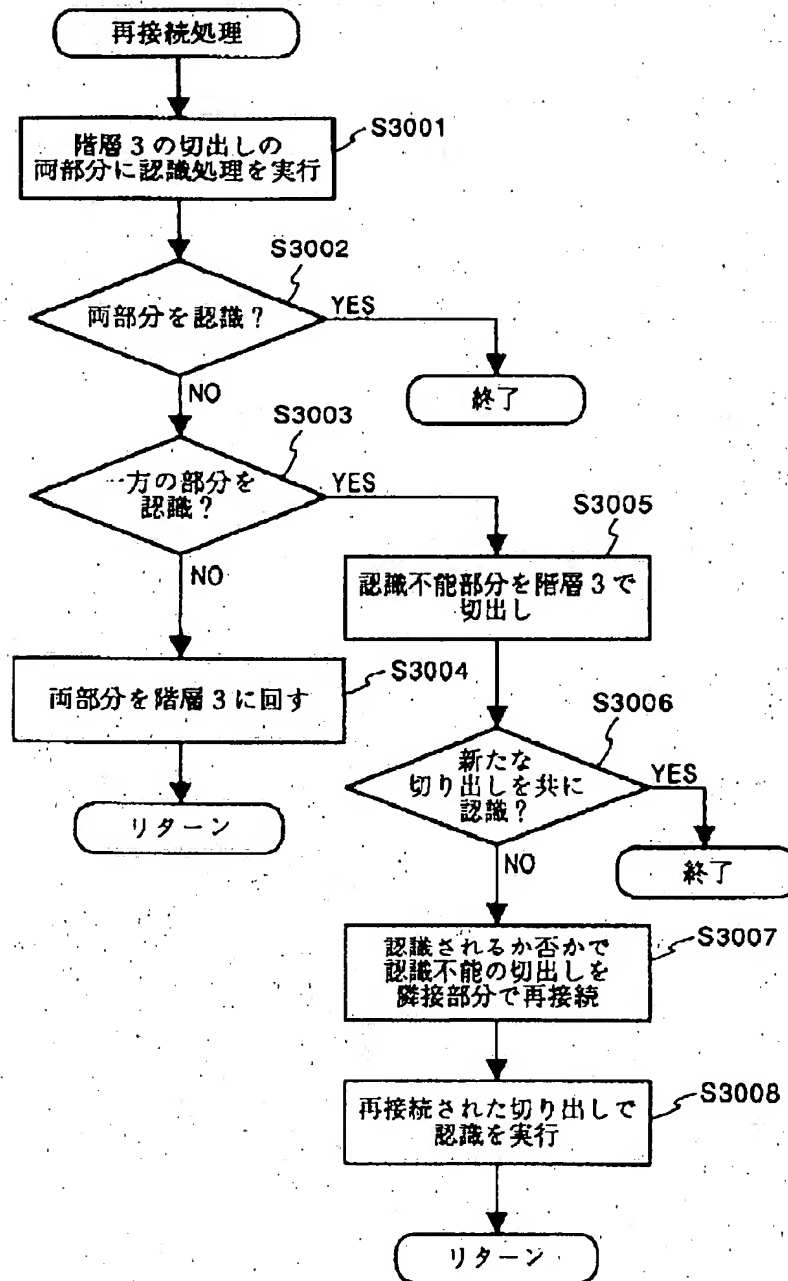
【図163】

```

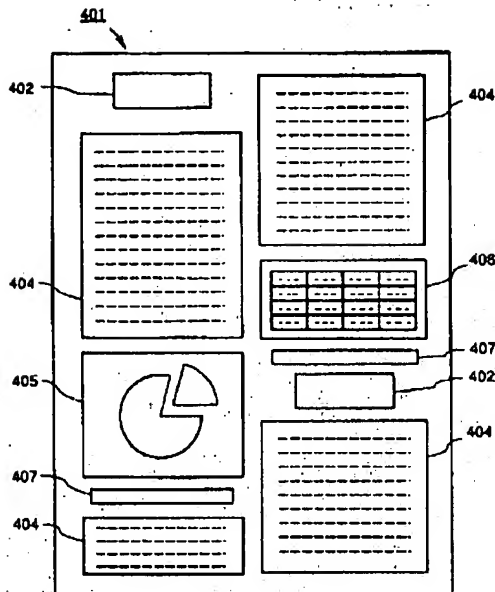
HISTOGRAM_THRESHOLD;
charon = charon+next; // Get the next character
free_ivector(histogram); // Deallocate the memory

```

【図30】



【図32】



【図44】

```

/* UNUSED */
static void onCallbackQuit(MessageBox(UxWidget, UxContext, UxCallbackArg)
{
    UxWidget widget;
    UxQuitDialog *UxContext;
    int UxCallbackArg;

    {
        void * UxThisWidget;

        UxThisWidget = UxWidgetToWidget(UxWidget);
        UxFromContext(UxContext);

        {
            exit(1);
        }

        UxToContext(UxContext);
    }
}

```

【図46】

```

_UxCon->DisplayForm = DisplayForm;
_UxCon->DisplayFormClose_FBG = DisplayFormClose_FBG;
_UxCon->DisplayForm_DrawArea = DisplayForm_DrawArea;
}

static void _UxFromContext(_UxCon)
{
    _UxDisplayForm = _UxCon;

    {
        DisplayForm = _UxCon->DisplayForm;
        DisplayFormClose_FBG = _UxCon->DisplayFormClose_FBG;
        DisplayForm_DrawArea = _UxCon->DisplayForm_DrawArea;
    }
}

/* UNUSED */
static void activateCallback_DisplayFormClose_FBG(UxWidget, UxContext, UxCallbackArg)
{
    UxWidget widget;
    _UxDisplayForm *UxContext;
    int UxCallbackArg;

    {
        void * UxThisWidget;

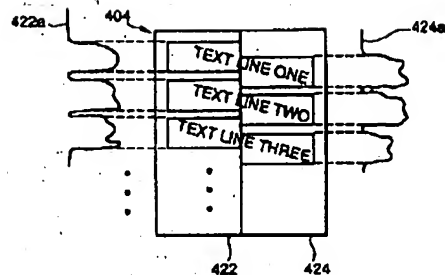
        UxThisWidget = UxWidgetToWidget(UxWidget);
        UxFromContext(UxContext);

        {
            UxPopdownInterface(DisplayForm);
        }

        UxToContext(UxContext);
    }
}

```

【図33C】



【図62】

```

int width, height;
int x, y, xdis, ydis, i;
static fid = 13;

init_graphics(wd);
xs_xr_via(wd);
xdis = ((height-10)/(TOTALCOLOR));
xs_setfont(wd);
for (i = 0; i < TOTALCOLOR; i++) {
    xs_set_foreground(color_code[i], wd);
    xs_setfont(wd, fid);
    xs_string(wd, 30, (i+1)*xdis+20, color_label[i]);
}
xs_flush(wd);

```

【図133】

```

#define R 0
#define G 1
#define B 2

int color_comp[3][3] = {{0x49, 0xc3, 0xc0},
                        {0xf0, 0x00, 0xc0},
                        {0xf0, 0x00, 0xf0},
                        {0xf0, 0xf0, 0xc0},
                        {0xf0, 0xf0, 0x00},
                        {0xf0, 0x00, 0x00},
                        {0x00, 0xf0, 0xf0},
                        {0x49, 0xc0, 0xf0},
                        {0x4b, 0xc0, 0xf0}},

/*
int color_comp[3][3] = {{0x00, 0x00, 0x00},
                        {0x00, 0x00, 0xc0},
                        {0x00, 0x00, 0x00},
                        {0x00, 0x00, 0xc0},
                        {0x00, 0x00, 0xc0},
                        {0x00, 0x00, 0xc0},
                        {0x00, 0x00, 0xc0},
                        {0x00, 0x00, 0xc0},
                        {0x00, 0x00, 0xc0}};
*/

```

【图 3 7】

[illegible]

—53—

[illegible]

—54—

[illegible]

【☒40】

[illegible]

[illegible]

【図42】

```

_UxCon->FileSelectionDialog = FileSelectionDialog;
_UxCon->FileSelectionBox = FileSelectionBox;
_UxCon->parent = parent;

static void _UxPromContext(_UxCon)
{
    _UxFileSelectionDialog = _UxCon;
    FileSelectionDialog = _UxCon->FileSelectionDialog;
    FileSelectionBox = _UxCon->FileSelectionBox;
    parent = _UxCon->parent;

/* ARGUSED */
static void cancelCallback_FileSelectionDialog(UxWidget, UxContext, UxCallbackArg)
{
    UxWidget widget;
    _UxFileSelectionDialog = _UxContext;
    int int;
    UxCallbackArg;

    widget = UxThisWidget;
    UxThisWidget = UxWidgetToWidget(UxWidget);
    _UxPromContext(UxContext);

    {
        UxPopdownInterface(FileSelectionDialog);
    }
    _UxToContext(UxContext);
}

/* ARGUSED */
static void okCallback_FileSelectionBox(UxWidget, UxContext, UxCallbackArg)
{
    UxWidget widget;
    _UxFileSelectionDialog = _UxContext;
    int int;
    UxCallbackArg;

    widget = UxThisWidget;
    UxThisWidget = UxWidgetToWidget(UxWidget);
    _UxPromContext(UxContext);

    {
        handle tiff_handle;
        UxPopdownInterface(FileSelectionDialog);
        strcpy(filename, get_input_filename());
        printf("Open (%s)\n", filename);
        tiff_handle = UxCreateSubproc("/usr/local/bin/X11/imageview", filename);
    }
    UxPopdownInterface();
    if (tiff_handle == -1) {
        error_create_subproc();
        return;
    }
    if (UxRunSubproc(tiff_handle, NULL) == -1) {
        error_run_subproc();
        return;
    }
    open_tiff_file(filename);
}
_UxToContext(UxContext);
}

```

【図48】

```

static void _UxPromContext(_UxCon)
{
    _UxInfoForm = _UxCon;
    InfoForm = _UxCon->InfoForm;
    InfoFormClose_PSD = _UxCon->InfoFormClose_PSD;
    InfoFormArea = _UxCon->InfoFormArea;

/* ARGUSED */
static void activateCallback_InfoFormClose_PSD(UxWidget, UxContext, UxCallbackArg)
{
    UxWidget widget;
    _UxInfoForm = _UxContext;
    int int;
    UxCallbackArg;

    widget = UxThisWidget;
    UxThisWidget = UxWidgetToWidget(UxWidget);
    _UxPromContext(UxContext);

    {
        UxPopdownInterface(InfoForm);
    }
    _UxToContext(UxContext);
}

```


[illegible]

[illegible]

【図49】

```

strcpy (log, name);
if ((pos = strchr(log, '\\')) != NULL)
    *pos = '\0';
return top;
}

/* ..... */
/* Return -1 if file cannot open */
/* ..... */
int check_file (name)
char *name;
{
    FILE *fp;

    if ((fp = fopen (name, "r")) == NULL)
        return -1;
    fclose (fp);
    return 1;
}

print_error (mess)
char *mess;
{
    printf ("***Error: %s\n", mess);
}

error_create_subproc ()
{
    printf ("***Error: Cannot create a subprocess!\n");
}

error_subproc_will_callback ()
{
    printf ("***Error: Cannot set subprocess exit callback!\n");
}

error_run_subproc ()
{
    printf ("***Error: Cannot start a subprocess!\n");
}

#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
#include <unistd.h>
#include <time.h>
#include <block.h>

struct timeval first, second, elapsed;
struct timespec top;

/* ..... */
/* Start the timing clock */
/* ..... */
start_clock ()
{
    gettimeofday (&first, 0);
}

/* ..... */
/* Stop the timing clock and return the elapsed time in milliseconds. */
/* ..... */
double stop_clock ()
{
    double time;

    gettimeofday (&second, 0);
    if (first.tv_usec > second.tv_usec) {
        second.tv_usec += 1000000;
        second.tv_usec -= first.tv_usec;
        elapsed.tv_usec = second.tv_usec - first.tv_usec;
        elapsed.tv_sec = second.tv_sec - first.tv_sec;
        time = (double)((double)elapsed.tv_usec / (double)1000);
        printf ("*** Elapsed time is %.3f\n msec.", time);
        return time;
    }

    /* ..... */
    /* Return the biasness of a file without the directory */
    /* ..... */
    char *extract_base_filename (name)
    char *name;

    char *p;
    p = strrchr(name, '\\');
    if (p == NULL)
        return pos;
    else
        return name;
}

/* ..... */
/* Return the biasness of a file */
/* ..... */
char *extract_base_filename (name)
char *name;
{
    char *p;
    p = strrchr(name, '\\');
    if (p == NULL)
        return pos;
    else
        return name;
}

```


[illegible]

—63—

[illegible]

[illegible]

【図53】

```

    printf("\nError in opening %s\n", filename);
    exit(1);
}
tagfile_position = 0;
/* Read in header and make sure it's here: */
i = getbyte(tagfile, tagfile_position);
/* printf("i = %d\n", i); */
if (i != 0x4949)
{
    printf("... Error: TIFF file not in Intel format!\n");
    exit(1);
}
/* Read in Magic number */
i = getbyte(tagfile, tagfile_position);
/* printf("i = %d\n", i); */
if (i != 0x002A)
{
    printf("... BAD Magic Number for TIFF file!\n");
    exit(1);
}
/* Find First File Directory */
i = getbytes(tagfile, tagfile_position,
             &tagfile_position + 1);
{
    i = getbyte(tagfile, tagfile_position);
}
/* Tags in Directory */
numtags = getbytes(tagfile, tagfile_position,
                    &tagfile_position + 1);
/* printf("numtags = %d\n", numtags); */
while (--numtags >= 0)
{
    type = getbyte(tagfile, tagfile_position);
    type = getbyte(tagfile, tagfile_position);
    len = getbytes(tagfile, tagfile_position,
                   &tagfile_position);
    value = getbytes(tagfile, tagfile_position,
                     &tagfile_position);
    if ((len != 1) && (tag & 32768) == 0)
    {
        printf("Error: Tag Length is %d\n", len);
        exit(1);
    }
    /* printf("tag = %d\n", tag); */
    switch (tag)
    {
        case 0x0000: /* NewSubFileType */
            if (value != 0)
            {
                printf("Can't deal with Non-Standard NewSubFileType\n");
                exit(1);
            }
            break;
        case 0x0001: /* NewSubFileType */
            if (value != 1)
            {
                printf("Subfile Type not full resolution\n");
                exit(1);
            }
            break;
        case 0x0002: /* Width */
            width = value;
            break;
        case 0x0003: /* Length */
            length = value;
            break;
    }
}

```

```

case 0x0003: /* Bits per sample */
    if (value != 3)
    {
        printf("Error: Bits/Sample must be 1!\n");
        exit(1);
    }
    break;
case 0x0003: /* Compression */
    if (value != 1)
    {
        printf("Compression Used! Error!\n");
        exit(1);
    }
    break;
case 0x0006: /* Black is */
    break;
case 0x0009: /* Thresholding tag */
    if (value != 1 || value != 3)
    {
        printf("Thresholding Tag = illegal value!\n");
        exit(1);
    }
    break;
case 0x0011: /* Offset to Aster Data */
    raster_offset = value;
    break;
case 0x0017: /* Rows */
    break;
case 0x0018: /* Rows */
    break;
case 0x0019: /* Rows */
    break;
case 0x001a: /* Rows */
    break;
case 0x001b: /* Rows */
    break;
case 0x001c: /* Rows */
    break;
case 0x001d: /* Rows */
    break;
case 0x001e: /* Rows */
    break;
case 0x001f: /* Rows */
    break;
case 0x0020: /* Rows */
    break;
case 0x0021: /* Rows */
    break;
case 0x0022: /* Rows */
    break;
case 0x0023: /* Rows */
    break;
case 0x0024: /* Rows */
    break;
case 0x0025: /* Rows */
    break;
case 0x0026: /* Rows */
    break;
case 0x0027: /* Rows */
    break;
case 0x0028: /* Rows */
    break;
case 0x0029: /* Rows */
    break;
case 0x002a: /* Rows */
    break;
case 0x002b: /* Rows */
    break;
case 0x002c: /* Rows */
    break;
case 0x002d: /* Rows */
    break;
case 0x002e: /* Rows */
    break;
case 0x002f: /* Rows */
    break;
case 0x0030: /* Rows */
    break;
case 0x0031: /* Rows */
    break;
case 0x0032: /* Rows */
    break;
case 0x0033: /* Rows */
    break;
case 0x0034: /* Rows */
    break;
case 0x0035: /* Rows */
    break;
case 0x0036: /* Rows */
    break;
case 0x0037: /* Rows */
    break;
case 0x0038: /* Rows */
    break;
case 0x0039: /* Rows */
    break;
case 0x003a: /* Rows */
    break;
case 0x003b: /* Rows */
    break;
case 0x003c: /* Rows */
    break;
case 0x003d: /* Rows */
    break;
case 0x003e: /* Rows */
    break;
case 0x003f: /* Rows */
    break;
case 0x0040: /* Rows */
    break;
case 0x0041: /* Rows */
    break;
case 0x0042: /* Rows */
    break;
case 0x0043: /* Rows */
    break;
case 0x0044: /* Rows */
    break;
case 0x0045: /* Rows */
    break;
case 0x0046: /* Rows */
    break;
case 0x0047: /* Rows */
    break;
case 0x0048: /* Rows */
    break;
case 0x0049: /* Rows */
    break;
case 0x004a: /* Rows */
    break;
case 0x004b: /* Rows */
    break;
case 0x004c: /* Rows */
    break;
case 0x004d: /* Rows */
    break;
case 0x004e: /* Rows */
    break;
case 0x004f: /* Rows */
    break;
case 0x0050: /* Rows */
    break;
case 0x0051: /* Rows */
    break;
case 0x0052: /* Rows */
    break;
case 0x0053: /* Rows */
    break;
case 0x0054: /* Rows */
    break;
case 0x0055: /* Rows */
    break;
case 0x0056: /* Rows */
    break;
case 0x0057: /* Rows */
    break;
case 0x0058: /* Rows */
    break;
case 0x0059: /* Rows */
    break;
case 0x005a: /* Rows */
    break;
case 0x005b: /* Rows */
    break;
case 0x005c: /* Rows */
    break;
case 0x005d: /* Rows */
    break;
case 0x005e: /* Rows */
    break;
case 0x005f: /* Rows */
    break;
case 0x0060: /* Rows */
    break;
case 0x0061: /* Rows */
    break;
case 0x0062: /* Rows */
    break;
case 0x0063: /* Rows */
    break;
case 0x0064: /* Rows */
    break;
case 0x0065: /* Rows */
    break;
case 0x0066: /* Rows */
    break;
case 0x0067: /* Rows */
    break;
case 0x0068: /* Rows */
    break;
case 0x0069: /* Rows */
    break;
case 0x006a: /* Rows */
    break;
case 0x006b: /* Rows */
    break;
case 0x006c: /* Rows */
    break;
case 0x006d: /* Rows */
    break;
case 0x006e: /* Rows */
    break;
case 0x006f: /* Rows */
    break;
case 0x0070: /* Rows */
    break;
case 0x0071: /* Rows */
    break;
case 0x0072: /* Rows */
    break;
case 0x0073: /* Rows */
    break;
case 0x0074: /* Rows */
    break;
case 0x0075: /* Rows */
    break;
case 0x0076: /* Rows */
    break;
case 0x0077: /* Rows */
    break;
case 0x0078: /* Rows */
    break;
case 0x0079: /* Rows */
    break;
case 0x007a: /* Rows */
    break;
case 0x007b: /* Rows */
    break;
case 0x007c: /* Rows */
    break;
case 0x007d: /* Rows */
    break;
case 0x007e: /* Rows */
    break;
case 0x007f: /* Rows */
    break;
case 0x0080: /* Rows */
    break;
case 0x0081: /* Rows */
    break;
case 0x0082: /* Rows */
    break;
case 0x0083: /* Rows */
    break;
case 0x0084: /* Rows */
    break;
case 0x0085: /* Rows */
    break;
case 0x0086: /* Rows */
    break;
case 0x0087: /* Rows */
    break;
case 0x0088: /* Rows */
    break;
case 0x0089: /* Rows */
    break;
case 0x008a: /* Rows */
    break;
case 0x008b: /* Rows */
    break;
case 0x008c: /* Rows */
    break;
case 0x008d: /* Rows */
    break;
case 0x008e: /* Rows */
    break;
case 0x008f: /* Rows */
    break;
case 0x0090: /* Rows */
    break;
case 0x0091: /* Rows */
    break;
case 0x0092: /* Rows */
    break;
case 0x0093: /* Rows */
    break;
case 0x0094: /* Rows */
    break;
case 0x0095: /* Rows */
    break;
case 0x0096: /* Rows */
    break;
case 0x0097: /* Rows */
    break;
case 0x0098: /* Rows */
    break;
case 0x0099: /* Rows */
    break;
case 0x009a: /* Rows */
    break;
case 0x009b: /* Rows */
    break;
case 0x009c: /* Rows */
    break;
case 0x009d: /* Rows */
    break;
case 0x009e: /* Rows */
    break;
case 0x009f: /* Rows */
    break;
case 0x00a0: /* Rows */
    break;
case 0x00a1: /* Rows */
    break;
case 0x00a2: /* Rows */
    break;
case 0x00a3: /* Rows */
    break;
case 0x00a4: /* Rows */
    break;
case 0x00a5: /* Rows */
    break;
case 0x00a6: /* Rows */
    break;
case 0x00a7: /* Rows */
    break;
case 0x00a8: /* Rows */
    break;
case 0x00a9: /* Rows */
    break;
case 0x00aa: /* Rows */
    break;
case 0x00ab: /* Rows */
    break;
case 0x00ac: /* Rows */
    break;
case 0x00ad: /* Rows */
    break;
case 0x00ae: /* Rows */
    break;
case 0x00af: /* Rows */
    break;
case 0x00b0: /* Rows */
    break;
case 0x00b1: /* Rows */
    break;
case 0x00b2: /* Rows */
    break;
case 0x00b3: /* Rows */
    break;
case 0x00b4: /* Rows */
    break;
case 0x00b5: /* Rows */
    break;
case 0x00b6: /* Rows */
    break;
case 0x00b7: /* Rows */
    break;
case 0x00b8: /* Rows */
    break;
case 0x00b9: /* Rows */
    break;
case 0x00ba: /* Rows */
    break;
case 0x00bb: /* Rows */
    break;
case 0x00bc: /* Rows */
    break;
case 0x00bd: /* Rows */
    break;
case 0x00be: /* Rows */
    break;
case 0x00bf: /* Rows */
    break;
case 0x00c0: /* Rows */
    break;
case 0x00c1: /* Rows */
    break;
case 0x00c2: /* Rows */
    break;
case 0x00c3: /* Rows */
    break;
case 0x00c4: /* Rows */
    break;
case 0x00c5: /* Rows */
    break;
case 0x00c6: /* Rows */
    break;
case 0x00c7: /* Rows */
    break;
case 0x00c8: /* Rows */
    break;
case 0x00c9: /* Rows */
    break;
case 0x00ca: /* Rows */
    break;
case 0x00cb: /* Rows */
    break;
case 0x00cc: /* Rows */
    break;
case 0x00cd: /* Rows */
    break;
case 0x00ce: /* Rows */
    break;
case 0x00cf: /* Rows */
    break;
case 0x00d0: /* Rows */
    break;
case 0x00d1: /* Rows */
    break;
case 0x00d2: /* Rows */
    break;
case 0x00d3: /* Rows */
    break;
case 0x00d4: /* Rows */
    break;
case 0x00d5: /* Rows */
    break;
case 0x00d6: /* Rows */
    break;
case 0x00d7: /* Rows */
    break;
case 0x00d8: /* Rows */
    break;
case 0x00d9: /* Rows */
    break;
case 0x00da: /* Rows */
    break;
case 0x00db: /* Rows */
    break;
case 0x00dc: /* Rows */
    break;
case 0x00dd: /* Rows */
    break;
case 0x00de: /* Rows */
    break;
case 0x00df: /* Rows */
    break;
case 0x00e0: /* Rows */
    break;
case 0x00e1: /* Rows */
    break;
case 0x00e2: /* Rows */
    break;
case 0x00e3: /* Rows */
    break;
case 0x00e4: /* Rows */
    break;
case 0x00e5: /* Rows */
    break;
case 0x00e6: /* Rows */
    break;
case 0x00e7: /* Rows */
    break;
case 0x00e8: /* Rows */
    break;
case 0x00e9: /* Rows */
    break;
case 0x00ea: /* Rows */
    break;
case 0x00eb: /* Rows */
    break;
case 0x00ec: /* Rows */
    break;
case 0x00ed: /* Rows */
    break;
case 0x00ee: /* Rows */
    break;
case 0x00ef: /* Rows */
    break;
case 0x00f0: /* Rows */
    break;
case 0x00f1: /* Rows */
    break;
case 0x00f2: /* Rows */
    break;
case 0x00f3: /* Rows */
    break;
case 0x00f4: /* Rows */
    break;
case 0x00f5: /* Rows */
    break;
case 0x00f6: /* Rows */
    break;
case 0x00f7: /* Rows */
    break;
case 0x00f8: /* Rows */
    break;
case 0x00f9: /* Rows */
    break;
case 0x00fa: /* Rows */
    break;
case 0x00fb: /* Rows */
    break;
case 0x00fc: /* Rows */
    break;
case 0x00fd: /* Rows */
    break;
case 0x00fe: /* Rows */
    break;
case 0x00ff: /* Rows */
    break;
}

```

【図54】

【図217】

Source Code for

Mono-Spaced ("Courier")
Segmentation

```

if ((v == EOF)
    printf("\n Error: Premature End of File found in tiff file\n");
    exit(1);
}
if ((v & 128) == black*127)
    page->pixel[i][j] = 0;
else
    page->pixel[i][j] = 1;
v = v << 1;
}
fclose(tagfile);
return page;

```

【図86】

```

    release_block(tmp1);
    tmp1 = tmp11;
}
for (; tmp2 != NULL; ) {
    tmp2 = tmp2->next;
    release_block(tmp2);
    tmp1 = tmp2;
}
free_rectblock(initial_block);
return;

/* release_block(root_rectblock); */

.....
/***** Draw text components for display *****/
.....
void block_circumulate(p, width, height, p_ori)
unsigned char **p;
int width, height;
unsigned char **p_ori;
{
    int i, j;
    /* open file for storing block data */
    original_p = p_ori;
    reduce_factor = SCALE_RATIO;

    for (i = 0; i <= 8; i++)
        for (j = 0; j <= 8; j++)
            direct_status[i][j] = 0;

    block_file = fopen("block_file.dat", "w");

    text_length = 150.0/SCALE_RATIO;
    total_text = 0;

    /* the root block stands for the whole page image unit */
    root_rectblock = new_block(0,0);
    root_rectblock->width = width;
    root_rectblock->length = height;
    root_rectblock->upper_y = 0;
    root_rectblock->lower_y = height-1;
    root_rectblock->left_x = 0;
    root_rectblock->right_x = width-1;

    /* find blocks inside the page range */
    block_in_range(p, 0, 0, width-1, height-1, root_rectblock);
    printf("get out of block_in_range\n");
    rearrange_rectblock(root_rectblock);
    printf("get out of rearrange_rectblock\n");

    text_discriminate(root_rectblock);

    printf("get out of text_disc\n");

    firstlevelnontextsearch(p, root_rectblock);

    printf("get out of firstlevelnontextsearch\n");
    print_block(1, root_rectblock);

    fclose(block_file);
    printf("before get out of block_construct\n");
}

```

```

..... FILE: BLOCK.MINE
.....
..... ***** AUTHOR: SUB-YEAR WING
..... ***** Date: 3-78-93
..... ***** Copyright 1993 Crown Information Systems
..... *****
.....
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include "blockmain.h"
include "blockgeneral.h"
include "images.h"

extern struct block_struct root_ptr_block;
struct linelock *filllinelock;
struct sectionlock *fillsectionlock;

extern unsigned char **control();
extern struct block_image *readcvt();
extern void block_circulate();
extern void init_line_lock();
extern void init_section_lock();
extern struct linelock *new_linelock();
extern struct sectionlock *new_sectionlock();
extern void print_linelock();
extern void print_sectionlock();
extern void free_linelock();
extern void free_sectionlock();

int size = 0;
struct block_image reduce_image;
struct linelock *line_lock;
float scale_displayform;
unsigned char "o_image";

/* unsigned char "block_matrix"; */

..... eliminate the noise
.....
void get_x_and_y_coordinates(float x, float y, second_x, second_y)
{
    int first_x, first_y, second_x, second_y;

    int i, j;

    for (i = first_x; i <= second_x; i++)
        for (j = first_y; j <= second_y; j++)
            pixel[i][j] = o_image[
                (pixel[i][j]-0)*164 +
                (pixel[j]-0)*164 +
                (pixel[i]-0)*164 +
                (pixel[j]-0)*164 +
                (pixel[i]-0)*164 +
                (pixel[j]-0)*164];
}

int main()
{
    .....
}

```


—69—

[illegible]

【図58】

```

for (temp1 = firstblock->firstblock, temp2 = NULL; ) {
    /* print first block of first block */
    temp2 = temp1->next;
    for (i = 0; i < temp1->numblock; i++) {
        if (temp1->numblock[i] != 0) {
            temp1->numblock[i] = temp1->numblock[i] + 1;
            temp1->numblock[i] = temp1->numblock[i];
        }
    }
    free(struct_valblock *temp1->numblock);
    free_valblock(temp1);
    /* print first free block of first block */
    temp1 = temp2;
    /* print first free block of first block */
    free_valblock(temp1);
}

/* ..... */
void free_section(firstsection)
struct sectionblock *firstsection;
{
    struct sectionblock *temp1, *temp2;
    struct block_valblock *temp1, *temp2;
    struct block_valblock *temp1, *temp2;

    print("get first free section\n");
    for (temp1 = firstsection->firstsection, temp2 = NULL; ) {
        temp2 = temp1->next;
        print("section = %d\n", temp1, temp1->index);
        for (temp1 = temp1->firstblock, temp2 = NULL; ) {
            temp2 = temp1->next;
            print("temp1 = %d line = %d first = %d current = %d\n", temp1,
                  temp1->index, temp1->firstblock->index,
                  temp1->current_block->index);
        }
        for (temp2 = temp1->firstblock, temp1 = NULL; ) {
            temp2 = temp1->next;
            free_block(temp2);
        }
        print("after free sec. temp1 = %d\n", temp1, temp1);
        temp1 = temp2;
        temp1 = temp2;
    }
    temp1->firstblock = NULL;
    print("free line temp1 = %d temp1->index = %d undered temp1->index\n",
          temp1, temp1->index, temp1->index, temp1->index);
    /* ..... */
    free(struct_valblock *temp1);
    print("after free line temp1 = %d\n", temp1, temp1);
}

```

[59]

```

    cap1 = NULL;
    cap2 = cap1;

    print("free section cap1 = %d\n", cap1);
    free(sectionblock "cap1");
    cap1 = NULL;
    cap2 = cap2;

    for (cap1 = firstsection->firstsection, cap1 != NULL, {
        print("current section = %d\n", cap1->index);
        cap1 = cap1->next;
        for (cap2 = cap1->firstlineblock, cap2 != NULL, {
            cap2 = cap2->next;
            print("line = %d first = %d\n", cap2->index, cap2->first->index);
            for (cap1 = cap2->firstblock, cap1 != NULL, {
                cap1 = cap1->next;
                free_block(cap1);
            }
            cap1 = cap1->next;
            print("next line cap1 = %d cap2->index = %d under %d cap2->index",
                  cap1, cap2->index, cap2->index->index, cap2->index->index);
            free(sectionblock "cap1");
            cap1 = NULL;
        }
        print("free line %d\n", cap1);
    }
    cap1 = cap1;
    cap2 = cap2;
    free_sectionblock = NULL;
    free_sectionblock = NULL;
    cap1 = NULL;
    print("free section %d\n", cap1);
    free_sectionblock "cap1";
    cap1 = cap1;
    free_sectionblock "firstsection";

    ..... Open tiff file and reduce the size by 3
    .....
    void open_tiff_file (filename)
    char filename[];

    print("step = %d\n", step);
    if (step != 0) { free_sectionblock "cap1", 0, reduce_image_height-1,

```

—72—

[illegible]

—73—

[図63]

```

.....
/* Allocate a float vector with range [m1..m2] */
float *vector(m1, m2)
{
    int m1, m2;

    /* Allocate a float vector with range [m1..m2] */
    float *v;
    v = (float *) malloc((m2-m1+1)*sizeof(float));
    if (!v) error("allocation failure in vector()");
    return v;
}

/* Allocate a double vector with range [m1..m2] */
double *vector(m1, m2)
{
    int m1, m2;

    /* Allocate a double vector with range [m1..m2] */
    double *v;
    v = (double *) malloc((m2-m1+1)*sizeof(double));
    if (!v) error("allocation failure in vector()");
    return v;
}

/* Allocate a character matrix with range [m1..m2] x [n1..n2] */
char **matrix(m1, m2, n1, n2)
{
    int m1, m2, n1, n2;

    /* Allocate a character matrix with range [m1..m2] x [n1..n2] */
    char **m;
    m = (char **) malloc((m2-m1+1)*sizeof(char *));
    if (!m) error("allocation failure 1 in matrix()");
    for (i = m1; i <= m2; i++)
        m[i] = (char *) malloc((n2-n1+1)*sizeof(char));
    if (!m[i]) error("allocation failure 2 in matrix()");
    return m;
}

/* Allocate an int matrix with range [m1..m2] x [n1..n2] */
int **matrix(m1, m2, n1, n2)
{
    int m1, m2, n1, n2;

    /* Allocate an int matrix with range [m1..m2] x [n1..n2] */
    int **m;
    m = (int **) malloc((m2-m1+1)*sizeof(int *));
    if (!m) error("allocation failure 1 in matrix()");
    for (i = m1; i <= m2; i++)
        m[i] = (int *) malloc((n2-n1+1)*sizeof(int));
    if (!m[i]) error("allocation failure 2 in matrix()");
    return m;
}

```

—75—

```

/* allocate a double matrix with range [n1, n2] [n1, n2] */
double **matrix(n1, n2, nch)
{
    int n1, n2, nch, ncl, ncl;

    int i;
    double **m;

    for (i = n1; i <= n2; i++) free((char*) (m[i]));

    free((char*) (m));

    /* allocate a double matrix */
    void free_matrix(n1, n2, nch, ncl);
    double **m;
    int n1, n2, nch, ncl;

    int i;
    for (i = n1; i <= n2; i++) free((double*) (m[i]));

    free((double*) (m));
}

/* allocate a float vector */
float *vector(n1, n2)
{
    int n1, n2;

    free((float*) (v));

    /* allocate an int vector */
    void free_vector(n1, n2);
    int *v, n1, n2;

    free((int*) (v));

    /* allocate a double vector */
    void free_double_vector(n1, n2);
    double *v;
    int n1, n2;

    free((double*) (v));

    /* allocate a float matrix */
    void free_matrix(n1, n2, nch, ncl);
    float **m;
    int n1, n2, nch, ncl;

    int i;
    for (i = n1; i <= n2; i++) free((float*) (m[i]));

    free((float*) (m));

    /* allocate an int matrix */
    void free_int_matrix(n1, n2, nch, ncl);
    int **m;
    int n1, n2, nch, ncl;

    int i;
    for (i = n1; i <= n2; i++) free((int*) (m[i]));

    free((int*) (m));
}

```

—76—

[illegible]

—77—

[illegible]

—78—

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

【図68】

```

void set_outline(n, y, nblock)
int n, y;
struct whiteblock *nblock;
{
    if (x < pairleft(0)) {
        if (pairright(0) == pairleft(0)) {
            tpair(numpair-2) = pairright(0);
            for (i = 0; i < pairright(0); i++) {
                if (i == pairright(0)) {
                    tpair(numpair-2) = pairright(0);
                    break;
                }
            }
        }
        else {
            tpair(numpair-2) = pairleft(0);
            tpair(numpair-1) = pairright(0);
            tpair(numpair-2) = pairleft(0);
            numpair--;
            break;
        }
        else {
            pairright(0) = pairleft(0) + numpair;
            for (i = 0; i < numpair; i++) {
                pairright(i) = tpair(i-1);
                pairleft(i) = tpair(i-1);
            }
            free((int *)numpair);
        }
    }
    ..... Search the direction of next "cor"
    .....
    void right_jump(p, old_direct, i, j, xl, yl, xd, yd, new_direct, new_x, new_y,
    unsigned char *p, location)
    int i, j, xl, yl, xd, yd;
    int new_direct, new_x, new_y;
    recording location;
    {
        int i, h;
        for (i = 0; i < search_jump[old_direct][0]; i++) {
            new_direct = search_jump[old_direct][i];
            new_x = search_jump[old_direct][1];
            new_y = search_jump[old_direct][2];
            if (p[new_y][new_x] == 0) {
                location = new_x;
                location = new_y;
                return;
            }
            new_x = 0;
            new_y = 0;
            location = 0;
        }
        ..... Set up the outline point for the searched white block
    }
}

```

```

void set_outline(n, y, nblock)
int n, y;
struct whiteblock *nblock;
{
    if (x < pairleft(0)) {
        if (pairright(0) == pairleft(0)) {
            tpair(numpair-2) = pairright(0);
            for (i = 0; i < pairright(0); i++) {
                if (i == pairright(0)) {
                    tpair(numpair-2) = pairright(0);
                    break;
                }
            }
        }
        else {
            tpair(numpair-2) = pairleft(0);
            tpair(numpair-1) = pairright(0);
            tpair(numpair-2) = pairleft(0);
            numpair--;
            break;
        }
        else {
            pairright(0) = pairleft(0) + numpair;
            for (i = 0; i < numpair; i++) {
                pairright(i) = tpair(i-1);
                pairleft(i) = tpair(i-1);
            }
            free((int *)numpair);
        }
    }
    ..... Search the direction of next "cor"
    .....
    void right_jump(p, old_direct, i, j, xl, yl, xd, yd, new_direct, new_x, new_y,
    unsigned char *p, location)
    int i, j, xl, yl, xd, yd;
    int new_direct, new_x, new_y;
    recording location;
    {
        int i, h;
        for (i = 0; i < search_jump[old_direct][0]; i++) {
            new_direct = search_jump[old_direct][i];
            new_x = search_jump[old_direct][1];
            new_y = search_jump[old_direct][2];
            if (p[new_y][new_x] == 0) {
                location = new_x;
                location = new_y;
                return;
            }
            new_x = 0;
            new_y = 0;
            location = 0;
        }
        ..... Set up the outline point for the searched white block
    }
}

```

```

void set_outline(n, y, nblock)
int n, y;
struct whiteblock *nblock;
{
    if (x < pairleft(0)) {
        if (pairright(0) == pairleft(0)) {
            tpair(numpair-2) = pairright(0);
            for (i = 0; i < pairright(0); i++) {
                if (i == pairright(0)) {
                    tpair(numpair-2) = pairright(0);
                    break;
                }
            }
        }
        else {
            tpair(numpair-2) = pairleft(0);
            tpair(numpair-1) = pairright(0);
            tpair(numpair-2) = pairleft(0);
            numpair--;
            break;
        }
        else {
            pairright(0) = pairleft(0) + numpair;
            for (i = 0; i < numpair; i++) {
                pairright(i) = tpair(i-1);
                pairleft(i) = tpair(i-1);
            }
            free((int *)numpair);
        }
    }
    ..... Search the direction of next "cor"
    .....
    void right_jump(p, old_direct, i, j, xl, yl, xd, yd, new_direct, new_x, new_y,
    unsigned char *p, location)
    int i, j, xl, yl, xd, yd;
    int new_direct, new_x, new_y;
    recording location;
    {
        int i, h;
        for (i = 0; i < search_jump[old_direct][0]; i++) {
            new_direct = search_jump[old_direct][i];
            new_x = search_jump[old_direct][1];
            new_y = search_jump[old_direct][2];
            if (p[new_y][new_x] == 0) {
                location = new_x;
                location = new_y;
                return;
            }
            new_x = 0;
            new_y = 0;
            location = 0;
        }
        ..... Set up the outline point for the searched white block
    }
}

```

[illegible]

【図70】

```

head->first->next->block = first;
break;
} else if (first->supp_y < tab->supp_y) continue;
else {
    first->prev = tab;
    first->next = tab->next;
    tab->next = first;
    if (first->next != NULL)
        first->next->prev = first;
    else
        head->current->next->block = first;
    break;
}
} else {
    if (head->first->next->block == NULL) {
        head->first->next->block = first;
        first->prev = first->next;
        head->current->next->block = first;
    } else {
        for (tab = head->current->next->block; tab != NULL; tab = tab->next->block) {
            if (tab->supp_y < first->supp_y) continue;
            first->next = head->first->next->block;
            head->first->next->prev = first;
            break;
        }
        else if (first->supp_y < tab->supp_y) continue;
        else {
            first->prev = tab;
            first->next = tab->next;
            tab->next = first;
            if (first->next != NULL)
                first->next->prev = first;
            else
                head->current->next->block = first;
            break;
        }
    }
}
}

void remove_block(struct first, head)
{
    struct block *first, *head;
    if (head->first->next->block == first) {
        head->first->next->block = first->next;
        if (first->next != NULL)
            first->next->prev = first->next;
        else
            head->current->next->block = first->next;
    } else {
        first->prev->next = first->next;
        if (first->next != NULL)
            first->next->prev = first->next;
        else
            head->current->next->block = first->next;
    }
}

void transfer_block(struct first, head)
{
    struct block *first, *originalhead, *head;
    first->next = first->next->next;
    if (first->next != NULL)
        first->next->prev = first->next;
    else
        head->current->next->block = first->next;
    first->next = first->next->prev;
    if (first->next != NULL)
        first->next->prev = first->next;
    else
        head->current->next->block = first->next;
}

void remove_block(struct first, head)
{
    struct block *first, *originalhead, *head;
    first->next = first->next->next;
    if (first->next != NULL)
        first->next->prev = first->next;
    else
        head->current->next->block = first->next;
}

void remove_block(struct first, head)
{
    struct block *first, *originalhead, *head;
    first->next = first->next->next;
    if (first->next != NULL)
        first->next->prev = first->next;
    else
        head->current->next->block = first->next;
}

```

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

-83-

—84—

[illegible]

【图 7-4】

[illegible]

【図75】

```

/* print out of block in range(n) */
..... count continuous ones and zeros
.....
void count_one_zero(block p, avg_one, avg_zero, std_one, std_zero)
{
    struct block_rect *block;
    unsigned char *op;
    float *avg_one, *std_one, *avg_zero, *std_zero;

    int i, j, N;
    int count, zero_count;
    int one, zero;
    int no_one, no_zero;

    printf("place 1\n");

    one = vector(p, block->length*block->width-1);
    zero = vector(p, block->length*block->width-1);
    no_one = 0; no_zero = 0;

    for (i = block->upper_y; i <= block->lower_y; i++) {
        one_count = 0;
        zero_count = 0;
        for (j = block->pairright[i][0]; j <= block->pairright[i][1]; j++) {
            x = block->pairleft[i][0];
            if (p[x][i] != 0) {
                one_count++;
                no_one++;
            } else {
                zero_count++;
                no_zero++;
            }
        }
        avg_one = (float)one_count / (float)no_one;
        std_one = sqrt((float)one_count / (float)no_one - (float)avg_one * (float)avg_one);
        avg_zero = (float)zero_count / (float)no_zero;
        std_zero = sqrt((float)zero_count / (float)no_zero - (float)avg_zero * (float)avg_zero);
    }

    free_vector(one, 0, block->length*block->width-1);
    free_vector(zero, 0, block->length*block->width-1);
    ..... count continuous ones and zeros
    .....
}

/* print out of block in range(n) */
.....
void count_one_zero(block p, avg_one, avg_zero, std_one, std_zero)
{
    struct block_rect *block;
    unsigned char *op;
    float *avg_one, *std_one, *avg_zero, *std_zero;

    int i, j, N;
    int count, zero_count;
    int one, zero;
    int no_one, no_zero;

    printf("place 1\n");

    one = vector(p, block->length*block->width-1);
    zero = vector(p, block->length*block->width-1);
    no_one = 0; no_zero = 0;

    for (i = block->upper_y; i <= block->lower_y; i++) {
        one_count = 0;
        zero_count = 0;
        for (j = block->pairright[i][0]; j <= block->pairright[i][1]; j++) {
            x = block->pairleft[i][0];
            if (p[x][i] != 0) {
                one_count++;
                no_one++;
            } else {
                zero_count++;
                no_zero++;
            }
        }
        avg_one = (float)one_count / (float)no_one;
        std_one = sqrt((float)one_count / (float)no_one - (float)avg_one * (float)avg_one);
        avg_zero = (float)zero_count / (float)no_zero;
        std_zero = sqrt((float)zero_count / (float)no_zero - (float)avg_zero * (float)avg_zero);
    }

    free_vector(one, 0, block->length*block->width-1);
    free_vector(zero, 0, block->length*block->width-1);
    ..... count continuous ones and zeros
    .....
}

/* print out of block in range(n) */
.....
void count_one_zero(block p, avg_one, avg_zero, std_one, std_zero)
{
    struct block_rect *block;
    unsigned char *op;
    float *avg_one, *std_one, *avg_zero, *std_zero;

    int i, j, N;
    int count, zero_count;
    int one, zero;
    int no_one, no_zero;

    printf("place 1\n");

    one = vector(p, block->length*block->width-1);
    zero = vector(p, block->length*block->width-1);
    no_one = 0; no_zero = 0;

    for (i = block->upper_y; i <= block->lower_y; i++) {
        one_count = 0;
        zero_count = 0;
        for (j = block->pairright[i][0]; j <= block->pairright[i][1]; j++) {
            x = block->pairleft[i][0];
            if (p[x][i] != 0) {
                one_count++;
                no_one++;
            } else {
                zero_count++;
                no_zero++;
            }
        }
        avg_one = (float)one_count / (float)no_one;
        std_one = sqrt((float)one_count / (float)no_one - (float)avg_one * (float)avg_one);
        avg_zero = (float)zero_count / (float)no_zero;
        std_zero = sqrt((float)zero_count / (float)no_zero - (float)avg_zero * (float)avg_zero);
    }

    free_vector(one, 0, block->length*block->width-1);
    free_vector(zero, 0, block->length*block->width-1);
    ..... count continuous ones and zeros
    .....
}

```

—87—

[illegible]

—88—

[illegible]

[図78]

```

tap->att = 0.000001;
if (tap->cover == COVER) {
    for (tap = tap->firstsubblock; tap != NULL; tap = tap->next) {
        if (tap->cover == DISCRIMINATE) {
            return tap->next;
        }
    }
}

tap = tap->next;
transfer_block(next, next->next, tap, headblock, headblock);
tap = tap;

else {
    samplelength = (long)tap->length;
    num = 1;
    tap->att = 1.0;
    tap = tap->next;
}

if (num > 0) {
    headblock->avg_weight = (float)samplelength/(float)num;
} else {
    headblock->avg_weight = 0.0;
}
return num;
}

/* print out of block in block */
void block_print(head, tap, gap, gap2)
int gap, gap2;
{
    struct block *newblock;
    int i, y;

    print("get into block with x = %d y = %d cover = %d first = %d last = %d",
        tap->firstsubblock,
        tap->lastsubblock);

    newblock = new_block(0, 0);
    b = tap->firstsubblock;

    newblock->pairright = b->pairright;
    newblock->pairleft = b->pairleft;
    newblock->upper_y = gap;
    newblock->lower_y = b->lower_y;
    newblock->right_x = b->right_x;
    newblock->left_x = b->left_x;
    for (y = newblock->upper_y; y < newblock->lower_y; y++) {
        for (i = 1; i < b->pairleft[y][0]; i++) {
            newblock->right_x = b->pairleft[y][1];
        }
        for (i = newblock->right_x; i < newblock->lower_y; i++) {
            for (y = newblock->pairleft[i][0]; y < newblock->lower_y; y++) {
                newblock->left_x = b->pairleft[y][1];
            }
        }
    }

    newblock->length = newblock->lower_y - newblock->upper_y;
    newblock->width = newblock->right_x - newblock->left_x;
    newblock->samplelength = newblock->length * newblock->width;
    newblock->att = gap;
}

if (newblock->att == FALSE) {
    newblock->att = TRUE;
}
print("before search block in block");
search_block_in_block(newblock, p);

/* print after search block in block */
if (newblock->firstsubblock != NULL) {
    newblock->firstsubblock->group = 2;
}

for (tap = newblock->firstsubblock; tap != NULL; tap = tap->next) {
    print("index = %d in group = %d", tap->index, tap->in_group);
}

/* rearrange newblock in block */
}

for (tap = newblock->firstsubblock; tap != NULL; tap = tap->next) {
    print("index = %d in group = %d", tap->index, tap->in_group);
}

if (newblock->pairright != NULL) {
    newblock->pairright->pairright = newblock->pairright->pairright;
    newblock->pairright->pairleft = newblock->pairright->pairleft;
}

print("act = train", newblock->att);
print("in");

return;
}

/* print out of block in block */
void block_print(head, tap, gap, gap2)
int gap, gap2;
{
    struct block *newblock;
    int i, y;

    print("get into block with x = %d y = %d cover = %d first = %d last = %d",
        tap->firstsubblock,
        tap->lastsubblock);

    newblock = new_block(0, 0);
    b = tap->firstsubblock;

    newblock->pairright = b->pairright;
    newblock->pairleft = b->pairleft;
    newblock->upper_y = gap;
    newblock->lower_y = b->lower_y;
    newblock->right_x = b->right_x;
    newblock->left_x = b->left_x;
    for (y = newblock->upper_y; y < newblock->lower_y; y++) {
        for (i = 1; i < b->pairleft[y][0]; i++) {
            newblock->right_x = b->pairleft[y][1];
        }
        for (i = newblock->right_x; i < newblock->lower_y; i++) {
            for (y = newblock->pairleft[i][0]; y < newblock->lower_y; y++) {
                newblock->left_x = b->pairleft[y][1];
            }
        }
    }

    newblock->length = newblock->lower_y - newblock->upper_y;
    newblock->width = newblock->right_x - newblock->left_x;
    newblock->samplelength = newblock->length * newblock->width;
    newblock->att = gap;
}

```


[illegible]

[illegible]

—93—

[illegible]

—96—

—97—

[illegible]

【例 8 8】

[illegible]

[圖89]

```

void add_block_into_line(rect_block, head_block, line_block)
struct block_rect *head_block, *line_block;
{
    struct block_rect *tail_block;

    remove_tail_block(&rect_block, head_block);
    head_block->next_block = NULL;
    if (line_block->first_block == NULL)
        line_block->first_block = rect_block;
    else
        line_block->last_block->next_block = rect_block;
    line_block->last_block = rect_block;
    line_block->upper_y = rect_block->upper_y;
    line_block->lower_y = rect_block->lower_y;
    line_block->sum_area_block +=
        rect_block->first_block->area_block + rect_block;
    line_block->max_x = line_block->first_block;
    rect_block->previous = NULL;
    if (rect_block->left_x < line_block->left_x)
        line_block->left_x = rect_block->left_x;
    if (rect_block->right_x > line_block->right_x)
        line_block->right_x = rect_block->right_x;
    if (rect_block->upper_y < line_block->upper_y)
        line_block->upper_y = rect_block->upper_y;
    if (rect_block->lower_y > line_block->lower_y)
        line_block->lower_y = rect_block->lower_y;
    line_block->sum_area_block +=
        rect_block->area_block;
}

/*..... test if a gap between two rect_block .....*/
int is_gap_block(rect_block, head_block, head_block)
{
    int count_x, i, count_y, count_z;
    int bound_x, bound_y;
    int len, maxcount, mincount;

    if (block->left_x == block->right_x) return 0;
    bound_x = min(block->upper_y, block->lower_y);
    bound_y = max(block->upper_y, block->lower_y);
    for (count_x = block->right_x; count_x < block->left_x; count_x--)
        for (i = bound_x; i <= bound_y; i++)
            if (original_x(i) == (count_x + 1))
                print("x", count_x, "y", count_y);
    count++;
    if (count1 == (bound-bound_y)) return 1;
}

len = (MAXPOINT/SCALE_RATIO + 1) * (SCALE_RATIO); MAXPOINT/SCALE_RATIO;
mincount = 0;
maxcount = 0;
}

```

【図90】

```

for (center_x = block->right_x; center_x < block->left_x; center_x--) {
    for (count = 0; 1 = bound(block)); lower_y; i++) {
        if (original_block[center_x][lower_y] != 0)
            count++;
    }
    if (count > maxcount) {
        maxcount = count;
        maxcount_x = center_x;
    }
}

if (maxcount == 1) {
    print("len = %d maxcount = %d x = %d y = %d\n", len, maxcount, maxcount_x,
        (bound(block) / 2));
    for (i = (bound(block) / 2); i < count; count++, i++) {
        original_block[i][maxcount_x] = 0;
    }
    return 1;
} else {
    return 0;
}

for (count = 0; i = upper; i = lower; i++) {
    if (text_align[center_x][count] != 0)
        print("x = %d upper_y = %d lower_y = %d count = %d\n",
            center_x, upper, lower, count);
    if (count <= cap_lower)
        return 1;
    else
        return 0;
}

/* ..... Drop block into line ..... */
void group_connected_block(stopx, column, current_lineblock, headblock, widthfill)
{
    struct lineblock *current_lineblock;
    struct block_ptr *headblock;
    struct block_ptr *widthfill;

    int x, count, i;
    int firstx, secondx;
    struct block_ptr *top_block;

    /* add the first block forward */
    add_block_into_line(widthfill, stopx, headblock, current_lineblock);
    for (i = stopx - 1; top_block = widthfill[stopx]; x = 0; x--) {
        if (widthfill[x] == top_block) {
            widthfill[x] = NULL;
            continue;
        }
    }

    count = 0;
    for (i = 0; i < count; i++) {
        if (widthfill[i] == NULL) continue;
        if (i <= 0 || i >= widthfill[i] - 1)
            for (i = 0; i < widthfill[i] - 1; i++) {
                if (count < widthfill[i])
                    count++;
            }
        else break;
    }
    if (count < widthfill[i] && (count > 0)) {
        count = 0;
        for (i = 0; i < count; i++) {
            if (widthfill[i] == NULL) continue;
            if (i <= 0 || i >= widthfill[i] - 1)
                for (i = 0; i < widthfill[i] - 1; i++) {
                    if (count < widthfill[i])
                        count++;
                }
            else break;
        }
    }
}

```


—102—

```

return 0;

if (capline->upper_y > max(firstline->lower_y, secondline->lower_y))
break;
else if (!LowerCap(capline->left_x, capline->right_x,
firstline->left_x, firstline->right_x,
secondline->left_x, secondline->right_x))
if (LowerCap(capline->left_x, capline->right_x,
secondline->left_x, secondline->right_x))
{
neighbor = 1;
break;
}
}

if (!neighbor)
if (!neighbor == 1)
{
if (LowerCap(secondline->upper_y, secondline->lower_y) <
firstline->upper_y, firstline->lower_y)
{
neighbor = 1;
break;
}
}
else if (LowerCap(secondline->left_x, secondline->right_x,
firstline->left_x, firstline->right_x,
secondline->left_x, secondline->right_x))
{
neighbor = 1;
break;
}
}

if (!neighbor)
if (!neighbor == 1)
{
if (LowerCap(secondline->upper_y, secondline->lower_y) <
firstline->upper_y, firstline->lower_y)
{
neighbor = 1;
break;
}
}
else if (LowerCap(secondline->left_x, secondline->right_x,
firstline->left_x, firstline->right_x,
secondline->left_x, secondline->right_x))
{
neighbor = 1;
break;
}
}

return 1;
}

return 0;
}

void sort_lines(firstlineblock)
struct lineblock *firstlineblock;
struct lineblock *cap;
int num, i, j, stop;

if (firstlineblock->firstlineblock == NULL return;
for (num = 0; cap = firstlineblock->firstlineblock; cap = cap->next)
{
if (num > 0) {
line_j = new_linepointer(0, num-1);
for (i = 0; cap = firstlineblock->firstlineblock; cap = cap->next;
line_j(i-1) = cap)
/* extra testing */
sort_linepointer(line_j, num);
firstlineblock->firstlineblock = line_j(0);
if (num > 1)
firstlineblock->current_subline = line_j(num-1);
}
}
}

```

```

for (sort_index = 0; sort_index < num; ++
    fill(fill_sentence, sorted_block(sort_index), firstblock, widthfill);
    ~ group a new line ~
    if (fill == TEXT) {
        apblock = new_line_block(firstblock);
        apblock->index = -1;
        group_connected_block(sort_index, column, apblock, firstblock,
            widthfill);
        continue;
    }
    else if (fill == SUBSTITUTION) {
        convert_block(sort_index, sort_index,
            firstblock, widthfill, sort_index);
        transfer_block(TEXT, TEXT, sorted_block(sort_index),
            firstblock, widthfill);
    }
    else {
        for (j = sort_index; j < num; ++j) {
            widthfill[j] = sorted_block(sort_index,
                sort_index);
        }
        sort_index = j;
    }
    for (i = 0; i < column; ++i) {
        if (widthfill[i] != NULL) {
            apblock = new_line_block(firstblock);
            apblock->index = -1;
            group_connected_block(sort_index, column, apblock, firstblock,
                widthfill);
            continue;
        }
        else {
            i = apblock->right;
        }
    }
    continue;
}
//***** group line for the block contain more than one group area *****
void line_group(firstblock, firstblock)
struct lineblock *firstblock;
struct lineblock *firstblock;
int i, x, y, j, center;
struct lineblock *apblock;
struct block_rect *apblock;
struct whiteblock *apwhite;
apblock = new_line_block(firstblock->group);
for (i = 0; i < firstblock->group; ++i)
    apblock[i] = new_line_block(firstblock);
for (apblock = firstblock->firstblock, apblock != NULL; i <
    numblock < apblock->next;
    firstblock = firstblock->next, apblock = apblock->next)
    apblock->next = firstblock;
}

```

[illegible]

—105—

【図96】

```

    for (i = 0; i <= (level-1)*2; i++)
        tapline = tapline->next();
    fprintf(line_file, " ");
    fprintf(line_file, "td content index = %d ", tapline->index);
    fprintf(line_file,
"xtd ytd l=td w=td att = %x block index=%d indexl=td g = %d l = %d r = %d\n",
        tapline->left_x, tapline->upper_y, tapline->length,
        tapline->width, tapline->att, tapline->first_block->index,
        tapline->current_block->index, tapline->group, tapline->caption,
        tapline->r_caption);
    print_n(nextlevel, tapline);
}

.....
..... Open the file for the output of line message .....
.....
void print_lineblock(firstline)
struct lineblock *firstline;
{
    line_file = fopen("line_file.dat", "w");
    print_ln(1, firstline);
    fclose(line_file);
}

```

【図118】

```

    if ((tapsection->att & (TITLE|HCL|ID|VOLUME|PICTURE))) continue;
    else
        init_section_block(tapsection, tapsection->firstlineblock, p,
            column);

.....
..... Print the line information into a file .....
.....
void print_sec(level, firstsec)
int level;
struct sectionblock *firstsec;
{
    int nextlevel, i;
    struct sectionblock *tapsubsec, *tapsec;
    nextlevel = level+1;
    for (tapsubsec = firstsec->firstsubsection; tapsubsec != NULL;
        tapsubsec = tapsubsec->next){
        for (i = 0; i <= (level-1)*2; i++)
            printf(" ");
        printf("index = %d ", tapsubsec->index);
        printf("xtd ytd l=td w=td att = %x\n",
            tapsubsec->left_x, tapsubsec->upper_y,
            tapsubsec->length, tapsubsec->width, tapsubsec->att);
    }

    for (tapsec = firstsec->firstmainsubsection; tapsec != NULL;
        tapsec = tapsec->next){
        for (i = 0; i <= (level-1)*2; i++)
            printf(" ");
        printf("content index = %d ", tapsec->index);
        printf("xtd ytd l=td w=td att = %x\n",
            tapsec->left_x, tapsec->upper_y, tapsec->length,
            tapsec->width, tapsec->att);
        print_sec(nextlevel, tapsec);
    }

.....
..... Open the file for the output of line message .....
.....
void print_secblock(firstsec)
struct sectionblock *firstsec;
{
    print_sec(1, firstsec);
}

```

【図137】

Source Code for Proportional Spaced Segmentation

—107—

[illegible]

【図98】

```

for (i = 0; cap < firstlineblock->firstsubline; cap += NULL; cap = cap->next)
    line_x[i++] = cap;
/* extra testing */
if (firstlineblock->currentsubline != NULL) {
    if (firstlineblock->currentsubline->index)
        printf("error: in sort_y\n");
    for (cap = firstlineblock->firstsubline; cap != NULL; cap = cap->next)
        line_x[i++] = cap;
    return line_x;
}

/*..... section block sorting .....*/
struct sectionblock *next_section(first; min)
int min;
{
    struct sectionblock *cap;
    struct sectionblock *line_x;
    int i, j, stop;

    min = 0;
    for (cap = first->firstsubline; cap != NULL; cap = cap->next)
        line_x[min++] = cap;

    for (i = 0; cap = first->firstsubline; cap != NULL; cap = cap->next)
        for (j = 0; j < min; j++)
            if (line_x[j]->upper > line_x[i]->upper) {
                ((line_x[j]->upper) > line_x[i]->upper) ?
                    ((line_x[j]->upper) > line_x[i]->upper) ?
                        cap = line_x[j];
                        line_x[j] = line_x[i];
                        line_x[i] = cap;
                        swap = 1;
                    }
                if (swap == 0) break;
            }
    return line_x;
}

/*..... Allocate new line block .....*/
struct sectionblock *new_sectionblock(int, int)
int n;
{
    struct sectionblock *v;

    v = (struct sectionblock *) malloc(sizeof(struct sectionblock));
    if (!v)
        printf("allocation error: section block\n");
    exit(1);
}

```

【図9】

```

if (current->firstlineblock == NULL)
    current->firstlineblock = line->next;
if (current->currentlineblock == line)
    current->currentlineblock = NULL;
else
    current->firstlineblock->previous = NULL;
    else if (current->currentlineblock == line->previous)
        current->currentlineblock = line->previous;
    else
        line->previous->next = line->next;
        line->next->previous = line->previous;
        line->next = NULL;
        line->next = NULL;

for ( ; ; )
{
    if (stop == 0) break;
    if (stop == 1)
    {
        stop = 0;
        if (current->firstlineblock == NULL)
            current->firstlineblock = line->next;
        else
            current->currentlineblock = line->previous;
        line->previous->next = line->next;
        line->next->previous = line->previous;
        line->next = NULL;
        line->next = NULL;
    }
    else
    {
        if (current->firstlineblock == line)
            current->currentlineblock = line;
        else
            current->currentlineblock = line->previous;
        line->previous->next = line->next;
        line->next->previous = line->previous;
        line->next = NULL;
        line->next = NULL;
    }
}

void put_section_line, cursection;
struct lineblock *line;
struct sectionblock *cursection;

if (current->firstlineblock == NULL)
    current->firstlineblock = line;
    line->previous = NULL;
    line->next = NULL;
    else
    {
        current->currentlineblock->next = line;
        line->previous = current->currentlineblock;
        line->next = NULL;
        line->next = NULL;
    }
}

// creates new section structure
// creates new section structure
struct lineblock *cur;
struct lineblock *cur;

struct sectionblock *s;
if (s == NULL)
    s = (struct sectionblock *) malloc(sizeof(struct sectionblock));
    if (s == NULL)
        printf("allocation error in sectionblock\n");
        exit(1);
    v->current = cur;
    v->previous = NULL;
    v->next = NULL;
    v->next = NULL;
    return v;

// creates new section structure
// creates new section structure
void sort_section_line, cur;
struct lineblock *line;
struct sectionblock *cur;

```


—110—

```

newhead->first_block->previous = line_train->current_block;
line_train->current_block->next = newhead->first_block;
newhead->first_block = line_train->first_block;
newhead->next_block = line_train->next_block;
if (line_train->first_block->next_block->next == NULL)
{
    if (line_train->right_x > newhead->right_x) line_train->left_x =
        newhead->right_x;
    if (line_train->upper_y > newhead->upper_y) line_train->right_y =
        newhead->upper_y;
    if (line_train->lower_y < newhead->lower_y) line_train->upper_y =
        newhead->lower_y;
    newhead->width = newhead->right_x - newhead->left_x;
    newhead->length = newhead->lower_y - newhead->upper_y;
}

//..... combination of line block backward
//.....
void b_line_combine(newhead, line_train)
struct lineblock *newhead, *line_train;
{
    newhead->current_block->next = line_train->first_block;
    line_train->first_block->previous = newhead->current_block;
    newhead->current_block = line_train->current_block;
    newhead->next_block = line_train->next_block;
    if (line_train->first_block->next_block->next == NULL)
    {
        if (line_train->right_x > newhead->right_x) line_train->left_x =
            newhead->right_x;
        if (line_train->upper_y > newhead->upper_y) line_train->right_y =
            newhead->upper_y;
        if (line_train->lower_y < newhead->lower_y) line_train->upper_y =
            newhead->lower_y;
        newhead->width = newhead->right_x - newhead->left_x;
        newhead->length = newhead->lower_y - newhead->upper_y;
    }
}

//..... line remove from one head and merge with another head
//.....
void line_remove(line_train, originalhead, newhead, act)
struct lineblock *line_train, *originalhead, *newhead;
{
    line_train->originalhead = line_train;
    while (line_train->next_block != NULL)
    {
        first(struct lineblock *) line_train;
    }
}

//..... cut if one line is inside another line
//.....
struct lineblock *inside(struct line, *secondline)
{
    if ((float)line->right_x < (float)line->upper_y, firstline->lower_y,
        (float)line->upper_y < (float)line->lower_y, secondline->right_x <
        secondline->length)
    {
        return 1;
    }
    return 0;
}

```

```

else if (cap->current->length == maxline->current->length)
    maxline = cap;
else if (cap->current->length < minline->current->length)
    minline = cap;
}

if (!leftlevel == 2) {
    if (!leftoverap(minline->current->left_x, minline->current->right_x))
        minline->current->width < minline->current->length-1;
    if (leftlength < ((float)minline->current->length)/3) {
        (maxline->current->length == minline->current->length ? 2 : 1)
        (maxline->current->width < minline->current->length - 2);
        leftlevel = 1;
    }
    if (nodestroy != minline) {
        line-transfer(maxline->current, head, maxline->current,
            minline->current);
        minline->current = NULL;
        linelist(minline->head) = NULL;
        minline = NULL;
    } else {
        line-transfer(head->current, head, maxline->current,
            maxline->current->next);
        maxline->current = NULL;
        linelist(maxline->head) = NULL;
        maxline = NULL;
    }
} else {
    if (leftlevel == 1)
        if (leftlevel == 1)
            minline->current->left_x < minline->current->left_x ?
                minline->current->length) :
                    minline->current->length;
            minline = NULL;
        } else {
            if (leftlevel == 1)
                line->connection = (maxline->NULL) minline-maxline;
            if (leftlevel == 2) leftlength /= (float)leftlevel;
            free((line *) mark-upper);
        }
    }
}

/*..... the number of "neighbor" on the right side .....*/
void test-right-neighbor(line, rightlevel, rightlength, head, linelist, nodestroy)
{
    int rightlevel;
    float rightlength;
    struct linelist *head, **linelist;
    struct maxline *nodelist;

```

【図102】

```

    else {
        lineTransfer(maxline-current-head, minline-current,
            maxline-current-act);
        maxline-current = NULL;
        minline = NULL;
    }
} else {
    rightlevel = 1;
    if (maxline-current-left < maxline-current-left-y) {
        rightmost = maxline-current-length;
        addln = NULL;
    } else {
        rightlength = maxline-current-length;
        addln = NULL;
    }
}

if (rightlevel == 1)
    line->connection = (maxline-act)/mainline-maxline;
if (rightlevel == 2) rightlength /= (float)rightlevel;
else((int *) main-support);

//----- text neighbor of some unknown block -----
void text_neighbor(line, leftlevel, rightlevel, leftlen, rightlen, first, lineList, nodist);
int leftlevel, rightlevel;
float leftlen, rightlen;
struct lineblock *first;
struct lineblock *nodist;
{
    text_neighbor(line, leftlevel, leftlen, first, lineList, nodist);
    text_neighbor(line, rightlevel, rightlen, first, lineList, nodist);
}

//----- Make title toward left direction -----
void left_title(currentline, line, first)
{
    struct lineblock *currentline;
    struct lineblock *line;
    struct lineblock *first;
    struct lineblock *nodist;
    float leftmost, leftlevel, leftlen;
    int combiner;

    leftmost = currentline-current-right;
    for (topline = currentline->connection; topline != NULL;
        topline = topline->connection; topline = currentline->connection)
        printf("capline %d currentline %d\n", topline-current-index,
            currentline-current-index);

    if (topline-current-left < rightmost-act) {
        text_neighbor(topline, leftlevel, leftlength, first, line, current);
        printf("capline %d leftlevel = %d\n",
            topline->connection-current-index, leftlevel);
    }
    if (leftlevel == 1) {
        if (cover_range(topline-current-support,
            topline-current-left-y, currentline-current-support,
            currentline-current-left-y) > 0.5) {

```

—113—

[illegible]

—114—

[illegible]

[図106]

```

segment(current_seg).begin_column = vector(i, columnset);
for (i = 0; i < columnset; i++)
    segment(current_seg).begin_column(i) = segment(i-1);
segment(current_seg).end_column(i) = segment(i);
for (j = 1; j <= segment(current_seg).columnset; j++)
    print("seg = 4d and = 1d", segment(current_seg).end_column(j));
print("\n");
for (topline = section(columnset).firstlineblock, topline = NULL;
    topline = topline->next;
    for (k = 0; k < columnset; k++)
        if ((topline->right_x <= segment(k-1).x)
            break;
        topline->right_x = segment(k).x;
        if (k < columnset)
            for (tap = section(k).firstlineblock, tap = NULL;
                tap = section(k).firstlineblock; tap->next; tap)
                if ((tap->right_x >= segment(k-1).x)
                    break;
                tap->right_x = segment(k).x;
                if (tap->right_x <= topline->right_x - text)
                    if (k < columnset)
                        tap->next = section(k+1).firstlineblock;
                    else
                        tap->next = section(columnset);
                tap = tap->next;
            }
        tap = tap->next;
    }
for (k = 0; k < columnset; k++)
    if ((topline->right_x <= segment(k-1).x)
        break;
    topline->right_x = segment(k).x;
    if (k < columnset)
        for (tap = section(k).firstlineblock, tap = NULL;
            tap = section(k).firstlineblock; tap->next; tap)
                if ((tap->right_x >= segment(k-1).x)
                    break;
                tap->right_x = segment(k).x;
                if (tap->right_x <= topline->right_x - text)
                    if (k < columnset)
                        tap->next = section(k+1).firstlineblock;
                    else
                        tap->next = section(columnset);
                tap = tap->next;
            }
        tap = tap->next;
    }
for (i = 0; i <= columnset; i++)
    print("section = 4d", i);

```

—117—

-118-

[illegible]

[図110]

```

    (top->right_x >= min(top->left_x, bottom->left_x))
    return 0;
}

// If overlap(top->upper_y, top->lower_y,
// min(top->upper_y, bottom->upper_y),
// max(top->lower_y, bottom->lower_y))
// min(top->left_x, top->right_x, bottom->left_x,
// min(top->left_x, bottom->left_x))
return 0;

// return 0;

// Test if some section fully inside the other section
// .....
int full_included(first, second)
struct section *first, *second;
{
    if ((first->left_x >= second->left_x) && (first->right_x <= second->right_x) &&
        (first->upper_y >= second->upper_y) && (first->lower_y <= second->lower_y))
        return 1;
    else
        return 0;
}

// .....
// Test if two section are neighbors (belong to same column)
// .....
int same_section(section, first, sec, num)
struct section *section;
int first, sec, num;
{
    int x1, y1, x2, y2, i, j, count;
    int in_section, in_column, in_section, in_column;
    struct section *top, *top2;
    int area1_x, area1_y, area1_x2, area1_y2;
    int area2_x, area2_y, area2_x2, area2_y2;
    char *;
    printf("In same section index 1: %d index 2: %d\n", section->first, section->second);
    sort_y(section->first);
    if (section->first->first_x < section->second->first_x)
        top = section->first;
    else
        top = section->second;
    if (!full_included(top->upper_y, top->lower_y,
        top->upper_y, top->lower_y) && 0) return 0;
    if ((top->section) && (top->section) && 0)
    {
        if (top->first->first_x <= top->second->first_x)
        {
            return 0;
        }
        else if (top->section)
        {
            return 0;
        }
    }
    printf("same section, same column\n");
    x1 = top->right_x;
    x2 = top->left_x;
    if (top->upper_y >= top->lower_y)
    {
        return 0;
    }
}

```


—123—

[illegible]

[illegible]

【116】

```

if ((tapt->firstlineblock) || (tapt->firstlineblock) ||
    combine == 2)
    print("boundary = %d\n", boundary(i)-index);
    current_seg = j;
    for (j = firstline; j < numline; j++)
        if (line_y_sort[j] <= line_y - tapsection->supper_y-1,
            nextfirst = j);
    if (boundary(i) <= line_y - tapsection->supper_y-1)
        at_blockidfirstline, lastline, line_y - tapsection->supper_y-1;
    if (firstline != nextfirst) {
        if (firstline <= line_y - tapsection->supper_y-1)
            subcurrentsec = firstlineblock;
        for (tapsection = subcurrentsec; tapsection != NULL;
            tapsection = tapsection->next) {
            tapsection->index = subcurrentsec;
            tapsection->width = tapsection->right_x - tapsection->left_x-1;
            tapsection->length = tapsection->lower_y - tapsection->upper_y-1;
            tapsection->boundary_index = i;
        }
    }
    if (boundary(i) <= line_y - tapsection->supper_y-1) {
        subcurrentsec = nextsection(firstsection);
        subcurrentsec->index = subcurrentsec;
        subcurrentsec->index = subcurrentsec;
    }
    subcurrentsec->firstlineblock = firstlineblock;
    subcurrentsec->supper_y = boundary(i)-supper_y;
    subcurrentsec->left_x = boundary(i)-left_x;
    subcurrentsec->right_x = boundary(i)-right_x;
    subcurrentsec->length = boundary(i)-length;
    subcurrentsec->width = boundary(i)-width;
    subcurrentsec->next = boundary(i)-next;
    boundary(i)-previous = NULL;
    boundary(i)-next = NULL;
}

print("nextfirst = %d\n", nextfirst, numline);
if (nextfirst <= numline) {
    current_seg = num_bound;
    firstline = nextfirst;
    lastline = num_bound;
    subcurrentsec = firstsection(firstsection, firstline, lastline);
    for (tapsection = subcurrentsec; tapsection != NULL;
        tapsection = tapsection->next) {
        tapsection->width = tapsection->right_x - tapsection->left_x-1;
        tapsection->length = tapsection->lower_y - tapsection->upper_y-1;
        tapsection->boundary_index = firstline;
    }
}

if (firstsection->firstsection != NULL) {
    print("firstsection = %d\n", firstsection->index);
    for (i = 0; i < num_bound; i++)
        print("segment = %d\n", i, segment(i), column, row);
}

```

【図117】

```

        for ( j = 1; j <= segment[i].column; j++)
            printf(" %d and = %d", segment[i].begin_column[j],
                segment[i].end_column[j]);
        printf("\n");
    }

    /* section_separate(firstsection); */
    printf("before section combine\n");
    /* section_combine(firstsection, boundary); */
    /* section_combine(firstsection, boundary); */
    inside_scut(firstsection);
    inside_scut(firstsection);

    /* free(firstsection.segment, 1); */
    free((struct lineblock *)firstsection);
    free((struct lineblock *)boundary);

    /* section_combine(firstsection, boundary); */
    /* inner_section_combine(firstsection); */

    /* ..... test ..... */
    void move_line_to_section(firstsection, firstline)
    struct sectionblock *firstsection;
    {
        struct lineblock *currentsection;
        int i;

        printf("get into move\n");
        for ( i = 0; tapline = firstline->firstline, tapline != NULL; )
        {
            currentsection = new_section(firstsection);
            currentsection->firstlineblock = currentsection->currentlineblock;
            currentsection->left_x = tapline->left_x;
            currentsection->right_x = tapline->right_x;
            currentsection->lower_y = tapline->lower_y;
            currentsection->upper_y = tapline->upper_y;
            currentsection->width = tapline->width;
            currentsection->length = tapline->length;
            currentsection->index = ++i;
            currentsection->next = NULL;
            firstsection->next = currentsection;
            tapline->previous = NULL;
            tapline = tapline->next;
        }

        printf("before order\n");
        for ( i = 0; tapline = firstline->firstline, tapline != NULL; )

```

【图 1 1 9】

[illegible]

—129—

[illegible]

[illegible]

—131—

[illegible]

[illegible]

【图 1 2 5】

```

.....
File: alloc_util.c .....
Author: Shin-Yuan Wang .....
Date : 1-28-92 .....
Copyright 1992 Canon Information Systems .....
.....

.....
test if covered range overlap .....
.....

int if_overlap(a1, a2, b1, b2)
{
    int a1, a2, b1, b2;

    if ((a2 <= b1) || (b2 <= a1)) return 0;
    return 1;
}

.....
minimum of two integers .....
.....

int min(a, b)
{
    int a, b;

    if (a <= b) return a;
    else return b;
}

.....
ratio of overlapping .....
.....

float cover_range(a1, a2, b1, b2)
{
    float a1, a2, b1, b2;

    float range = 0.0;
    int x1, x2;

    if ((a2 <= b1) || (b2 <= a1)) return range;
    else {
        x1 = max(a1, b1);
        x2 = min(a2, b2);

        range = ((float)(x2-x1+1)/(float)(a2-a1+1));
        return range;
    }
}

```

【图 1-3-1】

```

/* ..... */
/* ..... Filename: blockline.h ..... */
/* ..... Author: Shin-Yuan Wang ..... */
/* ..... Date: 1-15-93 ..... */
/* ..... Copyright 1993 Canon Information Systems ..... */
/* ..... */
#define EXPTT 0
#define OVERLAP 1
#define NEXT 2
#define SURROUNDIN 3
#define SURROUNDOUT 4

#define TH 1.7
#define WIDTHSTDY 1.3
#define CAPLENGTH 0

#define DOT_BOTTOM 0.8
#define DOT_WIDTH 1
#define LENDOVERLAP 0.5

#define LEFTCAP 0x10
#define RIGHTCAP 0x01
#define NOISE_WO 3

typedef int fillocondition;

extern void ka_draw_box();
extern void convert_to_wordblock();
extern void transfer_block();
extern void remove_textblock();
extern int max();
extern int min();
extern float cover_range();
extern struct blockline *new_linepointer();
extern void sort_linepointer();

```

```

.....Filename: block_main.h.....
.....Author: Shin-Yuan Wang.....
.....Date: 1-15-93.....
.....Copyright 1993 Canon Information System.....
.....
.....
#define TESTTIME 20
#define TEXT_BLOCK_BLOCK 0x21
#define LINEPICTURE_BLOCK 0x21
#define NPICTURE_BLOCK 0x21
#define TABLE_BLOCK 0x41
#define LINE_BLOCK 0x31
#define FRAME_BLOCK 0x51
#define UNKNOWN_BLOCK 0x71

#define TEXT_COLOR 0
#define TABLETEXT_COLOR 1
#define LINE_COLOR 2
#define PICTURE_COLOR 3
#define NPICTURE_COLOR 4
#define UNKNOWN_COLOR 5
#define TABLE_COLOR 6
#define FRAME_COLOR 7
#define LINEPICT_COLOR 8

enum {SP_READ = 0x01,
      SP_READUC = 0x02,
      SP_CURM = 0x04,
      SP_BLOCK = 0x08},

struct block_image
{
    int width;
    int height;
    unsigned char** pixel;
};

int koff(TESTTIME)[2] = {{0, 1}, {1, 2}, {0, 1}, {1, 2}, {0, 1},
                          {1, 2}, {0, 6}, {0, 0}, {1, 1}, {1, 1},
                          {2, 1}, {2, 2}, {1, 0}, {2, 1}, {1, 0},
                          {0, 1}, {0, 1}, {0, 1}, {1, 2}, {1, 2}};

int yoff(TESTTIME)[2] = {{0, 0}, {0, 0}, {1, 1}, {1, 1}, {2, 2},
                          {2, 2}, {0, 1}, {1, 2}, {0, 1}, {1, 2},
                          {0, 1}, {0, 2}, {0, 2}, {0, 1}, {1, 2},
                          {1, 2}, {1, 2}, {0, 1}, {1, 2}, {0, 1}};

#define TOTALCOLOR 7

char *color_label[]={
    "Text",
    "Text in the table",
    "Picture",
    "Line Drawing Graphics",
    "Solid line (Vertical or Horizontal)",
    "Table",
    "Frame"
};

int color_code[]={TEXT_COLOR, TABLETEXT_COLOR, NPICTURE_COLOR, PICTURE_COLOR,
                  LINE_COLOR, TABLE_COLOR, FRAME_COLOR};

```

```
/*.....*/  
/*.....*/ Filename: blocksection.h /*.....*/  
/*.....*/ Author : Shih-Yuan Wang /*.....*/  
/*.....*/ Date : 7-9-87 /*.....*/  
/*.....*/ Copyright 1992 Canon Information Systems /*.....*/  
/*.....*/  
  
#define GAP 16  
#define OVERLAP 0.8  
#define NOVERLAP 0.5  
#define LEVELOVERLAY 0.0  
#define LEFTSIDE 0x00  
#define LEFTALIGN 0x00  
#define LEFTTOPLONGER 0x00  
#define LEFTTOSHLONGER 0x20  
#define RIGHTSIDE 0x00  
#define RIGHTEALIGN 0x00  
#define RIGHTTOPLONGER 0x04  
#define RIGHTTOSHLONGER 0x02  
#define TEXTTEXT_LIKE 0x00  
  
#define VOICE_COUNT 8  
  
extern int line_inside();  
extern int maxl();  
extern int minl();  
extern int lf_overlap();  
extern void wa_draw_box();  
extern float cover_range();  
void exit();  
  
struct segment_map  
{ int column_no;  
int *begin_column;  
int *end_column;
```

—136—

[illegible]

—137—

[illegible]

【図135】

```

.....
..... Filename: table.h .....
..... Author: Shin-Yuan Wang .....
..... Date: 1-15-92 .....
..... Copyright 1992 Canon Information Systems .....
.....
#include "blockgeneral.h"

#define NONE 0
#define OFFSET 3
#define OFFSET2 4
#define NEW 1
#define OLD 2
#define NONSTOP 0
#define STOP 1
#define TABLE_TESTTIME 3

struct white_for_table
{
    int status;
    struct whiteblock *white;
    struct white_for_table *previous, *next;
};

```

【図136】

```

.....
..... Filename: images.h .....
..... Author: Shin-Yuan Wang .....
..... Date: 1-15-92 .....
..... Copyright 1992 Canon Information Systems .....
.....
#include<stdio.h>
#include<stdlib.h>

struct images
{
    int xsize;
    int ysize;
    int dpi;
    unsigned char **pixel;
};

```

【図142】

```

/*
 * Filename: autoinput.h
 * Header file for autoinput.c
 */
.....
/* Function Definitions for symalloc.c */
#ifndef CPP
/* Definition for C++ files */
extern "C" {
    void auto_initialize();
    int auto_input_int();
    void auto_terminate();
}
#else
/* Definition for C files */
void auto_initialize();
int auto_input_int();
void auto_terminate();
#endif

```

【図145】

```

.....
void zerovpp_terminate(nzlist)
    struct NonZeroVppList *nzlist; /* The list */
{
    free_vector(nzlist->list); /* Deallocate memory */
    nzlist->allocated = 0; /* Set so won't be accidently used */
    nzlist->size = 0;
}
.....
void zerovpp_add(nzlist, a, b)
    struct NonZeroVppList *nzlist; /* The list */
    int a, b; /* Start and stop of the zerovpp */
{
    if (nzlist->size) > nzlist->allocated {
        printf("NonZeroVppList FULL!\n");
        exit(1);
    }
    nzlist->list[(nzlist->size)++] = a;
    nzlist->list[(nzlist->size)++] = b;
}

```

【図244】

```

colors[x].flags = DoRed | DoGreen | DoBlue;
colors[x].red = the_color;
colors[x].green = the_color;
colors[x].blue = the_color;
the_color += 50;
}
XQueryColors(d, &def_colormap, colors, ncolors);
my_cmap = XCreateColormap(d, DefaultRootWindow(d), DefaultVisual(d, scr),
                          AllocAll);
XStoreColors(d, my_cmap, colors, ncolors);
}

```

[illegible]

(図140)

```

    temp3 = *(oc3++);
    *(nc++) = (temp1 & *(oc1++) | (temp2 & *(oc1++)) | (temp3 & *(oc1++));
    if (b) // Do the right end
    {
        *(nc) = *(oc1) | *(oc2) | *(oc3);
    }
    // Do the bottom line (if a == 1 of 2)
    nc = "nr";
    if (a == 1) // a == 1 it's one row
    {
        oc1 = "nr";
        for (j=0; j<neww; j++)
        {
            temp1 = *(oc1++);
            *(nc++) = temp1 & *(oc1++);
        }
        if (b) // Do the right end
        {
            *(nc) = *(oc1);
        }
        if (a == 2) // a == 2 it's two rows
        {
            oc1 = *(oc1++);
            oc2 = "nr";
            for (j=0; j<neww; j++)
            {
                temp1 = *(oc1++);
                temp2 = *(oc2++);
                *(nc++) = (temp1 & *(oc1++) | (temp2 & *(oc2++)));
            }
            if (b) // Do the right end
            {
                *(nc) = *(oc1) | *(oc2);
            }
        }
    }
}

oc1 = *(oc1++);
nc = "nr";
for (j=0; j<neww; j++)
{
    temp1 = *(oc1++);
    temp2 = *(oc2++);
    *(nc++) = (temp1 & *(oc1++) | (temp2 & *(oc2++)));
}
if (b) // Do the right end
{
    *(nc) = *(oc1) | *(oc2);
}
// Do the bottom line (if a == 1)
nc = "nr";
if (a == 1) // a == 1 it's one row
{
    oc1 = "nr";
    for (j=0; j<neww; j++)
    {
        temp1 = *(oc1++);
        *(nc++) = temp1 & *(oc1++);
    }
    if (b) // Do the right end
    {
        *(nc) = *(oc1);
    }
}

// L2 AND IN OF LOGIC
void tile_2x2x2x2(tilecmap* oldline, tilecmap* newline)
{
    int older, older; // size of old image
    int newer, newer; // size of new image
    unsigned char *oc1, *oc2, *oc3, *nc; // the old & new column pointers
    unsigned char *newc; // the new column pointer
    register unsigned char temp1, temp2, temp3;

    older = oldline->size;
    older = oldline->size; // Should be ==
    newer = newc->size; // Should be ==
    newline->realloc(neww, newh);

    a = older > newer? // How much is partially covered in the last row
    if (a == 1) // It's evenly aligned
    b = older < newer? // How much is partially covered in the last column
    if (b == 2)
    {
        newer--;
        b = 0;
        // == 0 for optimization of code below
    }
    oc = oldline->size;
    nc = newline->size;
    for (i=0; i<neww; i++) // Do the middle
    {
        oc1 = *(oc++);
        oc2 = *(oc++);
        oc3 = *(oc++);
        for (j=0; j<neww; j++)
        {
            temp1 = *(oc1++);
            temp2 = *(oc2++);
            temp3 = *(oc3++);
            *(nc++) = (temp1 & *(oc1++) | (temp2 & *(oc2++)) | (temp3 & *(oc3++)));
        }
        if (b) // Do the right end
        {
            *(nc) = *(oc1) | *(oc2) | *(oc3);
        }
    }
    // Do the bottom line (if a == 1)
    nc = "nr";
    if (a == 1) // a == 1 it's one row
    {
        oc1 = "nr";
        for (j=0; j<neww; j++)
        {
            temp1 = *(oc1++);
            *(nc++) = temp1 & *(oc1++);
        }
        if (b) // Do the right end
        {
            *(nc) = *(oc1);
        }
    }
    if (a == 2) // a == 2 it's two rows
    {
        oc1 = *(oc1++);
        oc2 = "nr";
        for (j=0; j<neww; j++)
        {
            temp1 = *(oc1++);
            temp2 = *(oc2++);
            *(nc++) = (temp1 & *(oc1++) | (temp2 & *(oc2++)));
        }
        if (b) // Do the right end
        {
            *(nc) = *(oc1) | *(oc2);
        }
    }
}

```


【図141】

```

/*
 * filename: autoinput.h
 * This file contains the code to input from a filename
 */
.....
#include <stdio.h>
#include "autoinput.h"
/* Global variables */

int auto_eof_detected;
FILE *auto_fp;
/* The file pointer */
int save_to_file;
/* Saving the input to a file */
int save_count;
/* Number of inputs before newline */
char filename[60];
/* The filename */
.....
void auto_initialize()
{
    int not_exit = 1;
    auto_eof_detected = 0;
    while(not_exit)
    {
        printf("Input filename for auto-input: ");
        scanf("%s", filename);
        auto_fp = fopen(filename, "r");
        if (auto_fp == NULL) /* Read open failed */
        {
            auto_fp = fopen(filename, "w");
            if (auto_fp == NULL) /* Write open failed */
            {
                printf("Can't open file \"%s\".", filename);
            }
            else /* Opened new file */
            {
                printf("New file \"%s\" opened as a new file.", filename);
                printf("Enter 1 at the next prompt.\n");
                save_to_file = 1;
                auto_eof_detected = 1;
                not_exit = 0;
            }
        }
        else /* Opening existing file failed */
        {
            not_exit = 0;
        }
    }
    printf("Do you wish to save the auto-input into this file? (1=yes) ");
    void auto_initialize();
    save_to_file = auto_input_int();
    printf("Warning: Change the 1st '1' to a '0' to turn off appending to the file.\n");
    save_count = 0;
}

int auto_input_int()
{
    int i;
    if (auto_eof_detected)
    {
        printf("\nManual input (-2 = newline, -9999 = exit) ");
    }
    scanf("%d", &i);
    if (i == -9999)
    {
        fclose(auto_fp);
        exit(0);
    }
    if (save_to_file == 1)
    {
        if (i == -1) /* Print new lines */
        {
            printf(auto_fp, "\n"); /* Print the new lines */
            printf(auto_fp, "\n");
            fflush(auto_fp);
            save_count = 0;
            return auto_input_int(); /* And input again */
        }
        printf(auto_fp, "%d ", i);
        printf(auto_fp, "\n");
        if (save_count > 3)
        {
            save_count = 0;
            printf(auto_fp, "\n");
        }
        return i;
    }
    if (strcmp(auto_fp, "td", 4) == EOF)
    {
        printf("\n===== END OF FILE DETECTED =====\n");
        if (save_to_file == 1)
        {
            fclose(auto_fp);
            auto_fp = fopen(filename, "a");
            if (auto_fp == NULL)
            {
                printf("===== WARNING: CANNOT APPEND TO FILE. See 'auto.h' filename =====\n");
                save_to_file = 0;
            }
            printf(auto_fp, "\n\n");
            auto_eof_detected = 1;
            return auto_input_int();
        }
        printf("\n\n");
        return i;
    }
}

```

[図143]

```

/* Plenum: Charlist.c
 * This file contains Charlist and Underdopline utility routines
 * .....
 * By Christopher Sherrick */
#include "charlist.h"
#include "malloc.h"

/* .....
 * Initialize and allocate a Charlist (set up curve pointers) */
void Charlist_Init(charlist, clist_max, vector_height)
struct Charlist *clist;
int clist_max;
int vector_height;
/* .....
 * A looping var */
int i;

/* Allocate memory for characters */
clist_max = (struct Character *) malloc((unsigned) clist_max
    * sizeof(struct Character));
if (clist_max == 0)
    printf("Memory Allocation Error in Charlist_Init(charlist)\n");
    exit(1);

/* Allocate memory for curves and set pointers */
clist_max = 0;
clist_max_allocated = clist_max; /* Max Allocated */
clist_max_free = 0; /* No first character */
clist_max_free_height = vector_height;

/* Allocate memory for curves and set pointers */
/* NOTE: The origin is between the array into the code */
for (i=0; i<clist_max; i++) {
    clist_max_free[i].curve = ivector(vector_height);
    clist_max_free[i].curve = ivector(vector_height);
}

/* Free a previously allocated Charlist */
void Charlist_Free(charlist)
struct Charlist *clist;
/* .....
 * Number of characters */
clist_max = 0;
/* No first character */
clist_max_free = 0;

/* Terminate deallocate the Charlist */
void Charlist_Deallocate(charlist)
struct Charlist *clist;
/* .....
 * for (i=clist_max_allocated-1; i>=0; i--) { /* Free curves */
    free_ivector(clist_max_free[i].curve);
    free_ivector(clist_max_free[i].curve);
}

```

```

/* .....
 * Place a new character in the Charlist */
/* Take the character beyond the charlist (charlist).
 * calculate loc, and insert it into the list */
void Charlist_Place_New_Char(charlist)
struct Charlist *clist;
/* .....
 * Pointer to the new character adding */
newchar = malloc(sizeof(struct Character)); /* Set newchar */
/* Calculate the position of the new character */
Charlist_Calculate_Location(newchar);

/* Check for full list */
/* increment the size of the list */
(clist_max++)
if (clist_max == clist_max_allocated) /* If too big.. */
    printf("Error in Charlist_Place_New_Char(): Character list is FULL!\n");
    exit(1);

/* Place new in the list character */
Charlist_Insert_Character(charlist, newchar);

/* This routine repositions a character in the charlist. It removes
 * the character, recalculates loc and inserts it in the list accordingly
 * void Charlist_Restore_Char(charlist, ch)
 * struct Charlist *clist;
 * struct Character *ch;
 * .....
 * Recalculate the position of the character */
Charlist_Restore_Location(ch);

if (clist_max < 1) /* If it's the only item in the list */
    return;

/* Remove the character from the list */
if (ch->prev == 0) /* If it's on list */
    clist_max_free = clist_max_free - 1;
else {
    clist_max_free = clist_max_free - 1;
    ch->prev = ch->next;
    if (ch->next == 0)
        clist_max_free = clist_max_free - 1;
}

/* Place the character in the list */
Charlist_Insert_Character(charlist, ch);

/* Insert a character after doing a out */

```

-144-

[illegible]

—145—

[illegible]

【図148】

```

// Copy clipped image
for (i=0; i<w2; i++)
    for (j=0; j<h2; j++)
        temp_pixel[i][j] = pixel[i][j];
Allocate temp_array, temp_array; // Copy temp image to current image
for (i=0; i<w2; i++)
    for (j=0; j<h2; j++)
        pixel[i][j] = temp_pixel[i][j];
}

// Read in picture file
void clip_image(FILE *f, int *w1, int *h1, int *w2, int *h2, int *c1, int *c2)
{
    FILE *tagfile; // Input file
    long int tagfile_position; // The our file position (in bytes)
    long int width; // Width of picture
    long int length; // Length of picture
    long int black; // Value of black pixel
    long int start; // Start of image data
    long int marker_offset; // Start of image data
    long int marker; // Tag values
    long int i, j;

    if ((tagfile = fopen(filename, "rb")) == NULL)
        error = "Error: TIFF file not in local format";
        exit(1);
    tagfile_position = 0;
    // Read in header and make sure it's local
    i = getbytes(tagfile, tagfile_position);
    if (i != 0x0001)
        error = "Error: TIFF file not in local format";
        exit(1);
    // Read in Magic number
    i = getbytes(tagfile, tagfile_position);
    if (i != 0x0001)
        error = "Error: TIFF file not in local format";
        exit(1);
    // Find first file directory
    i = getbytes(tagfile, tagfile_position);
    while (tagfile_position < 0)
    {
        i = getbytes(tagfile, tagfile_position);
        // Tag in Directory
        numtags = getbytes(tagfile, tagfile_position);
        while (numtags > 0)
        {
            tag = getbytes(tagfile, tagfile_position);
            type = getbytes(tagfile, tagfile_position);
            len = getbytes(tagfile, tagfile_position);
            value = getbytes(tagfile, tagfile_position);
            if ((len != 1) && ((tag & 32768) == 0))
                error = "Error: Tag length is 1/0";
        }
    }
}

```

【図149】

```

    } // while
    while (file_position < z_start_offset)
    {
        i = getbytes(tagfile, tagfile_position);
        //----- READ IN FILE -----
        allocate((int) length, (int) width);
        int vi;
        for (li = 0; li < length; li++)
        {
            for (j = 0; j < width; j++)
            {
                if ((j % 3) == 0)
                {
                    v = getbytes(tagfile);
                    if (v == 0)
                    {
                        cout << "Error: Premature End of File found in tiff file\n";
                        exit(1);
                    }
                    if ((v & 128) == black * 128)
                    {
                        pixel[li][j] = 0;
                    }
                    else
                    {
                        pixel[li][j] = 1;
                    }
                    v = v & 1;
                }
            }
        }
        fclose(tagfile);
        //-----
        // Read in a byte. All positions are equal
        // eoferror = 1 to error on EOF, 0 not
        int change = getbytes(FILE, long int file_position, int eoferror);
        int ci;
        ci = getc(f);
        if ((ci == EOF) && (eoferror == 1))
        {
            cout << "Error: Premature End of File (in getbytes)\n";
            exit(1);
        }
        file_position++;
        return ci;
    }
    //-----
    // Read in an integer from a file. (int, long)
    long int change = getbytes(FILE, long int ci);
    int ci;
    ci = getbytes(f, 1);
    ci = getbytes(f, 1);
    return ci * 0x100;
}
//-----
// Read in a long integer
// long int change = getbytes(FILE, long int ci);

```

```

    exit(1);
}
switch (tag)
{
    case 0x0002: // ImageWidthType
        if (value != 0)
        {
            cout << "Can't deal with non-standard non-integer type\n";
            exit(1);
        }
        break;
    case 0x0007: // ImageHeightType
        if (value != 1)
        {
            cout << "Subfile type not full resolution\n";
            exit(1);
        }
        break;
    case 0x0008: // Width
        width = value;
        break;
    case 0x0009: // Length
        length = value;
        break;
    case 0x0002: // Bits per sample
        if (value != 1)
        {
            cout << "Error: Bits/sample must be 1\n";
            exit(1);
        }
        break;
    case 0x0009: // Compression
        if (value != 1)
        {
            cout << "Compression Used! Error!\n";
            exit(1);
        }
        break;
    case 0x0006: // Black is
        black = value;
        break;
    case 0x0007: // Thresholding tag
        if ((value & 3) != (value > 3))
        {
            cout << "Thresholding tag = illegal value\n";
            exit(1);
        }
        break;
    case 0x0011: // Offset to Raster Data
        raster_offset = value;
        break;
    case 0x0017: // Type = value
        type = value;
        break;
    case 0x0018: // Break;
        break;
    case 0x0019: // Break;
        break;
    case 0x0012: // Break;
        break;
    default:
        if ((value & 31768) == 0)
        {
            cout << "!!!! Warning: An unsupported tag appears in the TIF file\n";
        }
        break;
} // switch

```


【图 152】

[illegible]

(図153)

```

for (i=c2-bot; i < c2-top; i++)
    botpos = c2-curved[i] + c1-curved[i];
}

// Compute position and exit
*d13 = botpos - toppos;

f = ((float) *d13) / ((float) width_small);
if (f < FMAX_VALID_DIST)
    return 1;
else
    return 0;
}

// This routine is called from comb_dist() with a pair of characters
// that probably isn't an i or j (i,j)
// Character. This determines if close enough to be combined.
// The first character
// The second character
// Which character is higher
// The last value
// The width of the smaller character
// Positions of the top and bottom
int i, botpos, toppos;
float f;

if (d13 == 1)
    // Case when c1 is higher than c2
    // Find bottom pos of c1
    if ((c1-bot - c1-stop) < 4)
        botpos = c1-stop;
    else
        botpos = 0;
    for (i=c1-bot; i < c1-stop; i++)
        botpos = c1-curved[i] + c1-curved[i];

// Find top pos of c2
if ((c2-bot - c2-stop) < 4)
    // If character very small, use .loc
    toppos = c2-stop;
else
    toppos = 0;
// Compute position and exit
*d13 = toppos - botpos;

f = ((float) *d13) / ((float) width_small);
if (f < FMAX_VALID_DIST)
    return 1;
else
    return 0;
}

// c2 is higher than c1
// Find top pos of c1
if ((c1-bot - c1-stop) < 4)
    // If character very small, use .loc
    botpos = c1-stop;
else
    botpos = 0;
for (i=c1-bot; i < c1-stop; i++)
    botpos = c1-curved[i] + c1-curved[i];

// Find bottom pos of c2
if ((c2-bot - c2-stop) < 4)
    // If character very small, use .loc
    toppos = c2-stop;
else
    toppos = 0;
}

```

```

else
    toprow = 0;
    for (i = ci - toprow; i < ci + toprow - 1; i++)
        toprow = ci - rowval(i) + ci - currow(i);
    }
    // Find bottom pos of c3
    if (i17 - botrow < ci - toprow) < 41
        botrow = ci - row; // If character very small, use .loc
    else
        botpos = 0;
        botpos = 0;
        for (i = c2 - bot - 4; i < ci - bot + 1; i++)
            botpos = ci - currow(i) + ci - currow(i);
        }
        // Compute position and exit
        // c2 = botpos - toprow;
        f = ((float) *d12) / ((float) width_small);
        if (f < PRAC_VALID_DIST)
            return 1;
        else
            return 0;
    }
    // This routine uses if larger characters are overlapping enough (left and
    // right) to cause a need for identifying type 1 percent signs.
    // Returns a 1 if overlapping for a type 1 percent, 0 otherwise.
    int overlap(int ci, int c1, int c2, int c3, int d12, int d13, int d14, int d15, int d16, int d17, int d18, int d19, int d20, int d21, int d22, int d23, int d24, int d25, int d26, int d27, int d28, int d29, int d30, int d31, int d32, int d33, int d34, int d35, int d36, int d37, int d38, int d39, int d40, int d41, int d42, int d43, int d44, int d45, int d46, int d47, int d48, int d49, int d50, int d51, int d52, int d53, int d54, int d55, int d56, int d57, int d58, int d59, int d60, int d61, int d62, int d63, int d64, int d65, int d66, int d67, int d68, int d69, int d70, int d71, int d72, int d73, int d74, int d75, int d76, int d77, int d78, int d79, int d80, int d81, int d82, int d83, int d84, int d85, int d86, int d87, int d88, int d89, int d90, int d91, int d92, int d93, int d94, int d95, int d96, int d97, int d98, int d99, int d100, int d101, int d102, int d103, int d104, int d105, int d106, int d107, int d108, int d109, int d110, int d111, int d112, int d113, int d114, int d115, int d116, int d117, int d118, int d119, int d120, int d121, int d122, int d123, int d124, int d125, int d126, int d127, int d128, int d129, int d130, int d131, int d132, int d133, int d134, int d135, int d136, int d137, int d138, int d139, int d140, int d141, int d142, int d143, int d144, int d145, int d146, int d147, int d148, int d149, int d150, int d151, int d152, int d153, int d154, int d155, int d156, int d157, int d158, int d159, int d160, int d161, int d162, int d163, int d164, int d165, int d166, int d167, int d168, int d169, int d170, int d171, int d172, int d173, int d174, int d175, int d176, int d177, int d178, int d179, int d180, int d181, int d182, int d183, int d184, int d185, int d186, int d187, int d188, int d189, int d190, int d191, int d192, int d193, int d194, int d195, int d196, int d197, int d198, int d199, int d200, int d201, int d202, int d203, int d204, int d205, int d206, int d207, int d208, int d209, int d210, int d211, int d212, int d213, int d214, int d215, int d216, int d217, int d218, int d219, int d220, int d221, int d222, int d223, int d224, int d225, int d226, int d227, int d228, int d229, int d230, int d231, int d232, int d233, int d234, int d235, int d236, int d237, int d238, int d239, int d240, int d241, int d242, int d243, int d244, int d245, int d246, int d247, int d248, int d249, int d250, int d251, int d252, int d253, int d254, int d255, int d256, int d257, int d258, int d259, int d260, int d261, int d262, int d263, int d264, int d265, int d266, int d267, int d268, int d269, int d270, int d271, int d272, int d273, int d274, int d275, int d276, int d277, int d278, int d279, int d280, int d281, int d282, int d283, int d284, int d285, int d286, int d287, int d288, int d289, int d290, int d291, int d292, int d293, int d294, int d295, int d296, int d297, int d298, int d299, int d300, int d301, int d302, int d303, int d304, int d305, int d306, int d307, int d308, int d309, int d310, int d311, int d312, int d313, int d314, int d315, int d316, int d317, int d318, int d319, int d320, int d321, int d322, int d323, int d324, int d325, int d326, int d327, int d328, int d329, int d330, int d331, int d332, int d333, int d334, int d335, int d336, int d337, int d338, int d339, int d340, int d341, int d342, int d343, int d344, int d345, int d346, int d347, int d348, int d349, int d350, int d351, int d352, int d353, int d354, int d355, int d356, int d357, int d358, int d359, int d360, int d361, int d362, int d363, int d364, int d365, int d366, int d367, int d368, int d369, int d370, int d371, int d372, int d373, int d374, int d375, int d376, int d377, int d378, int d379, int d380, int d381, int d382, int d383, int d384, int d385, int d386, int d387, int d388, int d389, int d390, int d391, int d392, int d393, int d394, int d395, int d396, int d397, int d398, int d399, int d400, int d401, int d402, int d403, int d404, int d405, int d406, int d407, int d408, int d409, int d410, int d411, int d412, int d413, int d414, int d415, int d416, int d417, int d418, int d419, int d420, int d421, int d422, int d423, int d424, int d425, int d426, int d427, int d428, int d429, int d430, int d431, int d432, int d433, int d434, int d435, int d436, int d437, int d438, int d439, int d440, int d441, int d442, int d443, int d444, int d445, int d446, int d447, int d448, int d449, int d450, int d451, int d452, int d453, int d454, int d455, int d456, int d457, int d458, int d459, int d460, int d461, int d462, int d463, int d464, int d465, int d466, int d467, int d468, int d469, int d470, int d471, int d472, int d473, int d474, int d475, int d476, int d477, int d478, int d479, int d480, int d481, int d482, int d483, int d484, int d485, int d486, int d487, int d488, int d489, int d490, int d491, int d492, int d493, int d494, int d495, int d496, int d497, int d498, int d499, int d500, int d501, int d502, int d503, int d504, int d505, int d506, int d507, int d508, int d509, int d510, int d511, int d512, int d513, int d514, int d515, int d516, int d517, int d518, int d519, int d520, int d521, int d522, int d523, int d524, int d525, int d526, int d527, int d528, int d529, int d530, int d531, int d532, int d533, int d534, int d535, int d536, int d537, int d538, int d539, int d540, int d541, int d542, int d543, int d544, int d545, int d546, int d547, int d548, int d549, int d550, int d551, int d552, int d553, int d554, int d555, int d556, int d557, int d558, int d559, int d560, int d561, int d562, int d563, int d564, int d565, int d566, int d567, int d568, int d569, int d570, int d571, int d572, int d573, int d574, int d575, int d576, int d577, int d578, int d579, int d580, int d581, int d582, int d583, int d584, int d585, int d586, int d587, int d588, int d589, int d590, int d591, int d592, int d593, int d594, int d595, int d596, int d597, int d598, int d599, int d600, int d601, int d602, int d603, int d604, int d605, int d606, int d607, int d608, int d609, int d610, int d611, int d612, int d613, int d614, int d615, int d616, int d617, int d618, int d619, int d620, int d621, int d622, int d623, int d624, int d625, int d626, int d627, int d628, int d629, int d630, int d631, int d632, int d633, int d634, int d635, int d636, int d637, int d638, int d639, int d640, int d641, int d642, int d643, int d644, int d645, int d646, int d647, int d648, int d649, int d650, int d651, int d652, int d653, int d654, int d655, int d65
```

—153—

—154—

[illegible]

combinational
combinational
combinational

[illegible]

—156—

[illegible]

—157—

[illegible]

【図160】

```

// Check for the dash between a character, and break it if needed
void cut_dash(
    image* line,          // The line
    struct Charlist* clist, // Pointer to charlist
    struct Character* ch,   // Pointer to the character we're on (modified
                           // To skip past newly inserted divisions)
    int* hist)             // The histogram (previously allocated)
{
    int p;

    // Reject if pixel width is too small.
    if (((ch->right - (ch->left) <= MIN_ABS_WIDTH_TO_CONSIDER)
        return;

//def PRINT_INFO
no_err_win();
print_character(line, "ch, 10, 10);
no_flush();

print("\nCharacter info:\n");
print("  top/bot = (%d, %d)\n", (ch->stop, (ch->bot);
print("  left/right = (%d, %d)\n", (ch->left, (ch->right);
endif

// Compute the histogram
histogram(line, ch, hist);

// Break left side
p = cut_left_dash(line, ch, hist);
if (p != 0)
{
    cut_character(line, clist, ch, p); // Cut the character
//def PRINT_INFO
print("Left dash cut\n");
print("\nCharacter info:\n");
print("  top/bot = (%d, %d)\n", (ch->stop, (ch->bot);
print("  left/right = (%d, %d)\n", (ch->left, (ch->right);
endif
}

// Break right side
p = cut_right_dash(line, ch, hist);
if (p != 0)
{
    cut_character(line, clist, ch, p); // Cut the character
//def PRINT_INFO
print("Right dash cut\n");
endif
}

//
// MAIN ROUTINE
//
// This will cut the dashes of of characters
void cut_dash(image* line, // The image
    struct Charlist* clist, // The current charlist
    int* histogram,         // The area of the histogram
    struct Character* charen, // The character we're processing

```

【図195】

```

/*
 * Filename: mymalloc.h
 * Header file for mymalloc.c
 */

/* Function definitions for mymalloc.c */
#ifdef CPP
/* Definition for C++ files */
extern "C" {
    int* ivector(int n);
    void free_ivector(int* v);
    float* fvector(int n);
    void free_fvector(float* v);
    int* ivector_ranged(int n, int m);
    void free_ivector_ranged(int* v, int m);
    int** matrix(int x, int y);
    void free_matrix(int** m, int x);
    char** cmatrix(int x, int y);
    void free_cmatrix(char** m, int x);
}
#else
/* Definition for C files */
/* The header file for malloc.c */
int* ivector();
void free_ivector();
int* fvector();
void free_fvector();
int** matrix_ranged();
void free_matrix_ranged();
int** matrix();
void free_matrix();
char** cmatrix();
void free_cmatrix();
#endif

```

【図223】

```

/*
 * Header file for mymalloc.h
 */
void zerovp_add(int* list, // The list
    struct Header* h, // The list
    int a, b); // Start and stop of the zerovp

if (list->size == list->allocated) {
    print("Warning: mymalloc FULL\n");
    exit(1);
}
list->list[list->size++] = a;
list->list[list->size++] = b;

```

【図16.1】

```

loc = loc_count + 0; // for position of the char on the fly
for (stop = 0; stop < 255; stop++)
{
    while (curvel[c] == curvel[stop]) // Loop until they're equal
    {
        if ((line_pixel[c] - curvel[stop]) < 0) // If there is a non empty space
            // to the right of the left border
            break;
        else
            curvel[c]++; // Slide the border over to meet it
    }
    while (curvel[c] < curvel[stop]) // Do the same with the right border:
    {
        if ((line_pixel[c] - curvel[stop]) > 0)
            break;
        else
            curvel[c]--;
    }
    if (curvel[c] != curvel[stop]) // there is a pixel on this line...
    {
        loc = curvel[c] + curvel[stop];
        loc_count++;
    }
    if (loc_count == 0)
    {
        loc = loc + 4 / loc_count; // compute the position (used for loc);
        pos = loc/8; // compute the real position
    }
    // find left and right
    left = curvel[stop];
    right = curvel[stop];
    for (stop = 0; stop < 255; stop++)
    {
        if (curvel[c] < curvel[stop])
            left = curvel[stop];
        if (curvel[c] > curvel[stop])
            right = curvel[stop];
    }
    if (loc == 0)
    {
        loc = 4 * (left + right);
        pos = (left + right) / 8;
    }
    for (stop = 0; stop < 255; stop++)
    {
        if (curvel[c] == curvel[stop])
            curvel[stop] = curvel[c] + pos;
    }
    // Copy all the new attributes in
    (*ch) - stop = loc;
    (*ch) - stop = top;
    (*ch) - bot = bot;
    (*ch) - left = left;
    (*ch) - right = right;
}

// This routine will cut each line two at point p. clet is updated,
// and ch returns as the 'on point to the right half of the character
// valid character
struct Character {
    struct Charlist *clet; // The character list
    struct Character *next; // The character to be cut
    int p; // Where to cut the character
};

// This routine will cut each line two at point p. clet is updated,
// and ch returns as the 'on point to the right half of the character
// valid character
struct Character {
    struct Charlist *clet; // The character list
    struct Character *next; // The character to be cut
    int p; // Where to cut the character
};

```

—160—

```

// char is the minimum value it
// left and right boundaries of where to search
int min, max;
int c;

// Compute the histogram
l31_histogram(hist, ch, hist);

p = ("ch" -> right);
while (p < p_end)
{
    // Start at left side
    // Exit when we've scanned the entire histogram
    // Looking for top-bottom crossing of threshold
    while (loop < p_end)
    {
        // histogram threshold
        hist[p] <= histogram_threshold;
        break;
    }
    // This is the left side of the area to search
    while (loop < p_end)
    {
        // Looking for bottom-top crossing of threshold
        if (hist[p] < histogram_threshold)
        {
            hist[p] > histogram_threshold;
            break;
        }
        // Remember right edge
        if (p < p_end)
        {
            // If we haven't gone over the edge
            // Search for min in 1,4,9 range
            min = hist[p];
            min_at = p;
            for (c = p-1; c <= p+1; c++)
            {
                if (hist[c] < min)
                {
                    min = hist[c];
                    min_at = c;
                }
            }
            // If the min is small enough, cut there
            if (min <= min_cut_threshold)
            {
                l31_cut_character(line, clist, ch, min_at); // Cut the character
                // Note: ch is changed to the right char
                // This should not mess things up
                printf("Level 3 made a cut: %d\n", min_at);
            }
        }
    }
}

// END ROUTINE
// This will do histogram touching on the characters for level 3 method 3
void level3_changes_line
{
    struct Character *clist; // The change
    struct Character *clist; // The current charlist
    int *histogram; // The area of the histogram
    struct Character *ch; // The character we're processing
    histogram = (vector(line.size)); // Allocate the memory
    char *clist = right; // Loop until end of list
    while (clist != 0)
    {
        l31_insert_breaks(line, clist, histogram, HD_CUT_THRESHOLD);
    }
}

```


【図165】

```

// Be initialized outside this routine
static Monitor* pList; // The line to process
int i;
int new_line; // Line to stop at
register char new_pixel_value;

//..... NO LOGIC ..... // The default
line_to_process = (input_line);

//..... THE OR LOGIC .....
if (logic_type == 1)
{
    if ((RESOLUTION_LINE_C_OR == 3) || (RESOLUTION_LINE_C_OR == 1))
    {
        if ((RESOLUTION_LINE_C_OR == 3) || (RESOLUTION_LINE_C_OR == 1))
        {
            // Using optimal or 2nd routine
            if (input_line < 1)
            {
                // The default
                line_to_process = (input_line);
            }
            else
            {
                // The standard
                new_line = (input_line - 1) / (RESOLUTION_LINE_C_OR);
                Process_line_resolution(new_line);
            }
        }
        else
        {
            // The standard
            new_line = (input_line - 1) / (RESOLUTION_LINE_C_OR);
            Process_line_resolution(new_line);
        }
    }
    for (i = 0; i < new_line; i++)
    {
        new_pixel_value = 0;
        new_line = (input_line - 1) / (RESOLUTION_LINE_C_OR);
        for (j = 0; j < new_line; j++)
        {
            new_pixel_value = new_pixel_value | input_line_pixel[j][i];
            Process_line_pixel(i, j);
        }
    }
    line_to_process = (Process_line);
}

//..... THE AND OR LOGIC .....
if (logic_type == 2)
{
    if ((RESOLUTION_LINE_C_AND == 3) || (RESOLUTION_LINE_C_AND == 1))
    {
        if ((RESOLUTION_LINE_C_AND == 3) || (RESOLUTION_LINE_C_AND == 1))
        {
            // Using optimal or 2nd routine
            if (input_line < 1)
            {
                // The default
                line_to_process = (input_line);
            }
            else
            {
                // The standard
                new_line = (input_line - 1) / (RESOLUTION_LINE_C_AND);
                Process_line_resolution(new_line);
            }
        }
        else
        {
            // The standard
            new_line = (input_line - 1) / (RESOLUTION_LINE_C_AND);
            Process_line_resolution(new_line);
        }
    }
    for (i = 0; i < new_line; i++)
    {
        new_pixel_value = 0;
        new_line = (input_line - 1) / (RESOLUTION_LINE_C_AND);
        for (j = 0; j < new_line; j++)
        {
            new_pixel_value = new_pixel_value | input_line_pixel[j][i];
            Process_line_pixel(i, j);
        }
    }
    line_to_process = (Process_line);
}
}

// Print a character on the screen
void Print_character(char c)
{
    struct Monitor* pList; // The position
    int x;
    int y;

    // The x starting position (for wrap around)
    int x_start;
    int x_end;
    int y_start;
    int y_end;

    // Number starting position
    for (i = 0; i < x_start; i++)
    {
        for (j = 0; j < y_start; j++)
        {
            if (i < x_start || j < y_start)
            {
                // Plot double line
                x_start = i;
                y_start = j;
            }
        }
    }
    x = x_start;
    y = y_start;

    // This will look at a segment. If it is separated from level 1 segmentation
    // and decide if it needs to go to level 2.
    int char_col; // The first column of the line to segment
    int char_row; // The first row of the line to segment
    int char_col; // The first column of the character is (line)
    int char_row; // The first row of the character is (line)

    // Delete following. Removes warnings during compilation
    int x = start_col + step_col + line_start;
    return 1;
}

// The image of the line to segment
// changes input_line.
// Output: The character list. Should
struct charlist *clist;

```

—163—

[illegible]

—164—

—166—

[illegible]

—167—

[illegible]

【図172】

```

peak = -1; // Set to not found
while ((i-pos) < 0) && (peak == -1))
    if (find_valley_to_right(proj, proj_size, ptx, max_valley_depth))
        peak = pos + 1;
return peak;

// Subroutine to find next valley to the right in a projection
// -1 = none found
int find_valley_to_right()
{
    int proj_size; // The projection
    int pos; // The max size of the projection (not inclusive)
    int max_valley_depth; // Highest point a valley can be
    int valley; // Where the valley is

    valley = -1; // Set to not found
    while ((i-pos) < proj_size) && (valley == -1))
        if (find_valley_to_left(proj, pos, max_valley_depth))
            valley = pos + 1;
    return valley;
}

// Subroutine to find next valley to the left in a projection
// -1 = none found
int find_valley_to_left()
{
    int proj_size; // The projection
    int pos; // The max size of the projection (not inclusive)
    int max_valley_depth; // Maximum value for the valley
    int valley; // Where the peak is

    valley = -1; // Set to not found
    while ((i-pos) < 0) && (valley == -1))
        if (find_valley_to_right(proj, pos, max_valley_depth))
            valley = pos - 1;
    return valley;
}

// Compute the Heurath's angle given a valley position
void find_valley_angle()
{
    int proj_size; // The projection vector
    int valley_pos; // The size of the projection vector
    double theta; // The position to the right of the last peak
    double theta; // Angle of left wall
    int min_peak_height; // Angle of right wall
    int max_valley_depth;

    int ptx, pty; // The coord of left peak
    int ptx, pty; // The coord of left valley bottom
    int ptx, pty; // The coord of right peak
    int ptx, pty;

    // Find pt: the peak to the left of the valley
    ptx = find_peak_to_left(proj, valley_pos, min_peak_height);
    if (ptx == -1) // If none found, make it the left side
        ptx = 0;
    pty = proj[pty]; // Get its height
}

```

```

// Find pt: the valley to the right of the given peak
ptx = find_valley_to_right(proj, proj_size, ptx, max_valley_depth);
if (ptx == -1) // If not found, make it the right side
    ptx = proj[ptx];
pty = proj[pty];

// Find pt: the next peak to the right of valley pt
pty = find_peak_to_right(proj, proj_size, ptx, min_peak_height);
if (pty == -1) // If not found, make it the right side
    pty = proj[pty];

// Find pt: the nearest valley to the left of peak at pt
ptx = find_valley_to_left(proj, ptx, max_valley_depth);
if (ptx == -1) // If not found, make it the left side
    ptx = 0;
pty = proj[pty];

// Find pt: the nearest valley to the left of peak at pt
ptx = find_valley_to_left(proj, ptx, max_valley_depth);
if (ptx == -1) // If not found, make it the left side
    ptx = 0;
pty = proj[pty];

// Heurath's optimal threshold finding routine
void compute_optimal_threshold()
{
    int proj_size; // The VTP (on array)
    int level; // The size of array
    int max_valley_depth; // The threshold level
    int min_peak_height; // The output
    int max_valley_pos; // The maximum value of the projection
    double a, b; // Values
    int i; // Index
    int j; // Index
    if ((i = fopen("constants.dat", "r")) == NULL)
        cerr << "Error in opening constants.dat\n";
    exit(1);

    fscanf(fp, "%f", &a); // Skip min char width
    for (i = 0; i < level; i++)
        fscanf(fp, "%f", &a); // Read a line
    if (i < level) // Exit
        return; // Return an error
    min_peak_height = -1;
    return;
}

// Close the file
fclose(fp);
// Print: *** LEVEL id = %f, min, level, a, b)
printf("LEVEL id = %f, min, level, a, b)",
    proj_size, a, b);
// Find maximum projection (for find peak and valley heights)
if (proj_size < 0)
    // Check for size < 0
    max_valley_depth = 0;
    min_peak_height = 0;
    return;
}

```

【図173】

```

break_position = (brk(0) + brk(image_size-1)) / 2;
// Insert the break
breaklist.add_break(breaklist, (break_position) * break_vector_height, break_position);
}

// Free memory
free_vector(breaklist);
free_vector(breaklist);
}

// Simple routine used by insert_angular_breaks
void insert_angular_breaks_min(
    int min, // The current minimum
    int max, // Where it's at
    double min_at, // The last minimum
    int min_at, // etc...
    double min_at, // etc...
    // Don't substitute if min_at == -1
    if (min_at < 0)
        return;
    if (min < min_at)
        min = min_at;
    min_at = min;
    min_at = min;
}

// Insert angular breaks in a segment
void insert_angular_breaks(
    // The breaklist
    // The starting break number
    // The stopping break number
    // Which set of thresholds to use
    // Projection vector and its size
    // Highest used value, and what value zero is
    // The maximum value of the projection
    // Position of valley
    // Maximum height for a valley
    // Minimum height for a peak
    // Angle of valley walls
    // Used for finding angular projections
    // A minimum and where it's located
    // A temporary min and where it's located
    break_count = 0;
}

// Allocate the projection vector
projection = Image::Alloc();
proj = Image::Alloc();
}

```

```

max_value_proj = proj(0);
for (i=1; i<image_size; i++)
    max_value_proj = proj(i);
max_value_proj = proj(0);

// Compute min and max valley heights
max_valley_depth = (int) (0.5 * (double) max_value_proj);
min_valley_height = (int) (0.5 * (double) max_value_proj);

// Reads in the median character width from the data file
int get_med_char_width(
    FILE *fp;
    double *w;
    if ((fp = fopen("constants.dat", "r")) == NULL)
        return 0;
    fscanf(fp, "%f", w);
    fclose(fp);
    return *w;
}

// Read min char width
fscanf(fp, "%f", w);
fclose(fp);
return *w;
}

// Insert angular breaks into the breaklist
void insert_angular_breaks(
    // The breaklist
    // The starting break number
    // The stopping break number
    // Which set of thresholds to use
    // Projection vector and its size
    // Highest used value, and what value zero is
    // The maximum value of the projection
    // Position of valley
    // Maximum height for a valley
    // Minimum height for a peak
    // Angle of valley walls
    // Used for finding angular projections
    // A minimum and where it's located
    // A temporary min and where it's located
    break_count = 0;
}

```

—170—

【図 176】

[illegible]

[図178]

```

// YNE
print(" Y N N YN");
print(" Combining Y N N Y and passing deeper");
breaklist_remove_break(blst, bn1);
breaklist_remove_break(blst, bn2);
b=2;
brk = breaklist_return_break_per(blst, bn);
brk-next_recog = 1;
brk-next_recog = 1;
return b;
}
if (blst[bn2] == 0 || (blst[bn2] == 1)) // YNN
{
    print(" Y N N YN");
    print(" Trying Y N N Y - ");
    if (pass_to_deeper_level(flag, blst, bn1, bn2));
    print(" Good");
    breaklist_remove_break(blst, bn2);
    breaklist_remove_break(blst, bn3);
    b=2;
    brk = breaklist_return_break_per(blst, bn);
    brk-next_recog = 1;
    return b;
}
print("badn Trying Y N N Y - ");
if (pass_to_deeper_level(flag, blst, bn1, bn2));
{
    print(" Good");
    breaklist_remove_break(blst, bn2);
    breaklist_remove_break(blst, bn3);
    b=2;
    brk = breaklist_return_break_per(blst, bn);
    brk-next_recog = 1;
    return b;
}
print("badn Trying Y N N Y - ");
if (pass_to_deeper_level(flag, blst, bn, bn-1));
{
    print(" Good");
    breaklist_remove_break(blst, bn-2);
    breaklist_remove_break(blst, bn-3);
    breaklist_remove_break(blst, bn-2);
    return b;
}
print("badn Combining Y N N Y and passing to deeper level");
breaklist_remove_break(blst, bn-2);
breaklist_remove_break(blst, bn-3);
brk = breaklist_return_break_per(blst, bn);
brk-next_recog = 1;
return b;
}
if (blst[bn] == 1) // We have YN
{
    print(" Y N YN");
    print(" Trying Y N Y - ");
    if (pass_to_deeper_level(flag, blst, bn-1, bn-3));
    {
        print(" Good");
        breaklist_remove_break(blst, bn-2);
        brk = breaklist_return_break_per(blst, bn-3);
        brk-next_recog = 1;
        return b;
    }
}
}

```

—175—

[illegible]

—178—

[illegible]

—179—

[illegible]

[illegible]

—182—

[illegible]

[illegible]

```

for (angle = range_t1_min; angle < range_t1_max; angle++)
{
    find_min_valley_at_theta(image, ch, angle, pl,
        min, min_at, min_theta);
}

// Now compute the valley
find_min_valley_at_theta(image, valley_pos, min_valley_depth,
    pl, val, val_at, val_theta);

// BEGIN PLOTTING ***
// Plot the tick mark under valley found
for (c = 0; c < pl; c++)
{
    range_t1_tick = vpp_min, current_vpp_tickval();
    range_t2_tick = vpp_min, current_vpp_tickval();
    range_t3_tick = vpp_min, current_vpp_tickval();
    range_t4_tick = vpp_min, current_vpp_tickval();
    range_t5_tick = vpp_min, current_vpp_tickval();
    range_t6_tick = vpp_min, current_vpp_tickval();
    range_t7_tick = vpp_min, current_vpp_tickval();
    range_t8_tick = vpp_min, current_vpp_tickval();
    range_t9_tick = vpp_min, current_vpp_tickval();
    range_t10_tick = vpp_min, current_vpp_tickval();
}

// END PLOTTING ***

// If we've done this vpp before, get results. If not, compute them
if (find_min_valley_at_theta(image, valley_pos, min_valley_depth,
    pl, val, val_at, val_theta) == 0)
{
    find_min_valley_at_theta(image, valley_pos, min_valley_depth,
        pl, val, val_at, val_theta);
}

// Print ("valley angles = %d, %d", theta1, theta2);

range_t1_min = theta1; // theta1 = 3.6
range_t1_max = theta2; // theta2 = 3.6
range_t2_min = theta1; // theta1 = 3.6
range_t2_max = theta2; // theta2 = 3.6
range_t3_min = theta1; // theta1 = 3.6
range_t3_max = theta2; // theta2 = 3.6
range_t4_min = theta1; // theta1 = 3.6
range_t4_max = theta2; // theta2 = 3.6
range_t5_min = theta1; // theta1 = 3.6
range_t5_max = theta2; // theta2 = 3.6
range_t6_min = theta1; // theta1 = 3.6
range_t6_max = theta2; // theta2 = 3.6
range_t7_min = theta1; // theta1 = 3.6
range_t7_max = theta2; // theta2 = 3.6
range_t8_min = theta1; // theta1 = 3.6
range_t8_max = theta2; // theta2 = 3.6
range_t9_min = theta1; // theta1 = 3.6
range_t9_max = theta2; // theta2 = 3.6
range_t10_min = theta1; // theta1 = 3.6
range_t10_max = theta2; // theta2 = 3.6

// Check ranges
range_t1_min = NOT_UNDER_RANGE;
range_t1_max = NOT_UNDER_RANGE;
range_t2_min = NOT_UNDER_RANGE;
range_t2_max = NOT_UNDER_RANGE;
range_t3_min = NOT_UNDER_RANGE;
range_t3_max = NOT_UNDER_RANGE;
range_t4_min = NOT_UNDER_RANGE;
range_t4_max = NOT_UNDER_RANGE;
range_t5_min = NOT_UNDER_RANGE;
range_t5_max = NOT_UNDER_RANGE;
range_t6_min = NOT_UNDER_RANGE;
range_t6_max = NOT_UNDER_RANGE;
range_t7_min = NOT_UNDER_RANGE;
range_t7_max = NOT_UNDER_RANGE;
range_t8_min = NOT_UNDER_RANGE;
range_t8_max = NOT_UNDER_RANGE;
range_t9_min = NOT_UNDER_RANGE;
range_t9_max = NOT_UNDER_RANGE;
range_t10_min = NOT_UNDER_RANGE;
range_t10_max = NOT_UNDER_RANGE;

// (range_t1_min = range_t1_min) // If overlap
{
    print(("Range reduced, %d", (id1) to ", range_t1_min,
        range_t1_max, range_t1_min, range_t1_max);
    // Then combine & do one search
    range_t1_min = range_t1_min;
    range_t1_max = range_t1_max;
    print(("Reduced to ", range_t1_min, range_t1_max);
}

// Search the angle range and find the best breakpoint the range
min = vpp(2);
min_at = 0;
min_theta = 0;
for (i = 0; i < 100; i++)
{
    min = vpp(2);
    min_at = i;
}

// Search rest of the
for (angle = range_t1_min; angle < range_t1_max; angle++)
{
    find_min_valley_at_theta(image, ch, angle, pl,
        min, min_at, min_theta);
}

```

—185—

[illegible]

```

charlist_remove_char(list, c);
c = c_prev;
charlist_recalculate_pos(char);

// Recalculate location

// This routine works with l1_poscount, comb_right() to check '99's after
// successful recognition. I.e. if M(99) Y tries M (M(99) Y), which
// will detect and remove some errors caused by a false successful recognition
int l1_poscount_recognize()
{
    struct charlist *list;
    struct Character *ch;
    struct Character *rch;

    // The character to start from
    // Leftmost character to search (exclusive)

    struct Character *c;
    // The leftmost character of the range
    // The 1st char removed

    int ret_value;
    int temp_value;

    ret_value = temp_value = 0;
    c = ch;

    while(1)
    {
        c = c_prev;
        temp_value--;
        // Get the previous character
        // At edge
        if (c == lch)
        {
            return ret_value;
        }
        // Return if not a M
        if (c == 'M')
        {
            return ret_value;
        }
        // Return if not '99'
        if (c == '99')
        {
            return ret_value;
        }
        // Recombine (flag, c, ch)
        // Combine it into one character (in c)
        l1_combine_char(list, c, ch);
        c = c_prev;
        temp_value--;
        ret_value = temp_value;
        ch = c;
    }
}

// This routine will start at character ch and search to left from ch
// grouping characters to see if they're recognized. If so they are
// combined. Note: ch can == 0.
int l1_poscount_comb_left()
{
    struct charlist *list;
    struct Character *ch;
    struct Character *rch;

    // The character to start from
    // Leftmost character to search (exclusive)

    struct Character *c;
    // The leftmost character of the range
    // The 1st char removed

    int ret_value = 0;
    c = ch;

    while(1)
    {
        // This loop is "return"ed out of
        if (c == '99')
        {
            return ret_value;
        }
        // If recognition (flag, c, ch) // c to ch recognized?
        l1_combine_char(list, c, ch);
        // Combine it into one character (in c)
        l1_combine_char(list, c, ch);
        c = c_prev;
        temp_value--;
        ret_value = temp_value;
        ch = c;
    }
}

```

【圖 191】

[illegible]

【図192】

```

//
// =====
void level4(
    cimage line,          // The line
    struct Charlist *clist) // The charlist
{
    int proj_min, proj_max, // Used to compute size of proj
    int i;

    #ifdef VISIBLE
    xa_clr_vwin();
    int c;
    Cursor1_X = Cursor1_Y = 10;
    line_plot(Cursor1_X, Cursor1_Y); // Plot input image
    Cursor2_Y = line.sizeX + 10; // Final output goes at Cursor2
    Cursor2_X = Cursor1_X;
    Cursor2_Y = Cursor2_Y - line.sizeX + 10;
    Cursor3_X = Cursor2_X;
    for (c=0; c<WINDOW_SIZE_X; c++)
        xa_plotxy(c, Cursor3_Y); // Plot horiz line
    Cursor3_Y = Cursor2_Y + 10;
    Cursor4_Y = Cursor3_Y;
    Cursor4_X = Cursor3_X;
    #endif

    if (line.sizeX > MAX_CHAR_HEIGHT)
    {
        printf("\n*** ERROR ***\n");
        printf("Line size (%d) is greater than max line size allocated in Multilevel\n",
            line.sizeX);
        printf("***** routines (MAX_CHAR_HEIGHT = %d). Increase MAX_CHAR_HEIGHT, MAX_C\n",
            MAX_HEIGHT);
        exit(1);
    }

    // Allocate Proj and VPP
    i = (NOT_MAX_RANGE/NOT_DEC_RESOLUTION)-1;
    proj_min = -rec((line.sizeX));
    proj_max = line.sizeX - proj_min;
    proj = ivector_ranged(proj_min, proj_max);
    VPP = ivector(line.sizeX);
    VPP_Zero = 1; // Erase the cache

    /* ..... Program goes here ..... */
    goodcut(line, clist);

    #ifdef VISIBLE
    // Print final character list
    xa_plot_charlist(line, clist, Cursor2_X, Cursor2_Y);
    #endif
    printf("Displaying final level 4 charlist. Enter 0 to continue? ");
    // print_and_charlist(clist, clist->first); // ***** DELETE ME *****
    i = auto_input_int();

    // Deallocate
    free_ivector(VPP);
    free_ivector_ranged(proj, proj_min);
}

```

【図203】

```

//
// The routines in this file determine if a character is recognized or not
// (for segmentation)
// =====
#include "xa.h" // Include Rick's graphics routines
#include "autoinput.h"

extern int WINDOW_SIZE_X;

int recognized(a, size_r, size_c) /* Return a 0 for not recognized, 1 for
    recognized */
{
    int **a; // The character matrix a[row][col]
    int size_r; // The number of rows in a
    int size_c; // The number of columns in a

    int r,c; // The row and column of the image
    int dummy; // A dummy variable

    #ifdef VISIBLE
    // This routine will print the char at the screen's bottom left corner
    for (r=0-10; r<size_r+20; r++) // Loop through the rows
        for (c=0-10; c<size_c+20; c++) // Loop through the cols
            if ((r >= 0) && (r < size_r) && (c >= 0) && (c < size_c))
                if (a[r][c] != 0) // If the pixel is on
                    xa_plotxy(c+10, r + WINDOW_SIZE_Y - size_r - 10);
                else
                    xa_unplotxy(c+10, r + WINDOW_SIZE_Y - size_r - 10);
                xa_unplotxy(c+10, r + WINDOW_SIZE_Y - size_r - 10);
    xa_flush(); // Flush the display buffer (to make sure
                // the character gets printed on the screen)
    #endif

    printf("Character displayed. Enter return value (1 = good, 0 = bad) ");
    dummy = auto_input_int();
    return dummy;
}

```

【図193】

```

/* This program contains random vector and matrix mallocs */
#include <stdio.h>
#include <stdlib.h>
/* Routine for error during memory allocation */
void error(char *s) /* The error text */
{
    if (*s) printf("%s\n", s);
    printf("Memory allocation error (SUBPROGRAM ABORTED: %s)\n", s);
    exit(1);
}

/* ===== VECTOR ===== */
/* Allocate an integer vector of size (0..n-1) */
int *vec(int n) /* Size of vector */
{
    int *v;
    v = (int *) malloc(sizeof(int) * n);
    if (!v) allocation_error("in vector()");
    return v;
}

/* Free an integer vector v of size (0..n-1) */
void free_vector(int *v, int n) /* The vector */
{
    free((int *) v);
}

/* ===== VECTOR ===== */
/* Allocate an float vector of size (0..n-1) */
float *fvec(int n) /* Size of vector */
{
    float *v;
    v = (float *) malloc(sizeof(float) * n);
    if (!v) allocation_error("in fvector()");
    return v;
}

/* Free an float vector v of size (0..n-1) */
void free_fvector(float *v, int n) /* The vector */
{
    free((float *) v);
}

/* ===== RANGED VECTOR ===== */
/* Allocate a ranged vector v of size (n..m-1) */
int *rvec(int n, int m) /* The range */
{
    int *v;
    v = (int *) malloc(sizeof(int) * (m-n));
    if (!v) allocation_error("in rvector_ranged()");
    return (v+n);
}

/* Free an integer ranged vector v of size (n..m-1) */
void free_rvec(int *v, int n, int m) /* The starting index of v */
{
    free((int *) (v-n));
}

/* ===== MATRIX ===== */
/* Allocate an integer matrix of size (0..r-1)(0..c-1) */
int **mat(int r, int c) /* The size of the matrix */
{
    int **m;
    m = (int **) malloc(sizeof(int *) * r);
    for (i=0; i<r; i++) {
        m[i] = (int *) malloc(sizeof(int) * c);
        if (!m[i]) allocation_error("in matrix()");
    }
    return m;
}

/* Free an integer matrix m of size (0..r-1)(0..c-1) */
void free_matrix(int **m, int r, int c) /* The matrix */
{
    for (i=0; i<r; i++)
        free((int *) m[i]);
    free((int **) m);
}

/* ===== MATRIX ===== */
/* Allocate an char matrix of size (0..r-1)(0..c-1) */
char **cmat(int r, int c) /* The size of the matrix */
{
    char **m;
    m = (char **) malloc(sizeof(char *) * r);
    for (i=0; i<r; i++) {
        m[i] = (char *) malloc(sizeof(char) * c);
        if (!m[i]) allocation_error("in cmatrix()");
    }
    return m;
}

/* Free an char matrix m of size (0..r-1)(0..c-1) */
void free_cmat(char **m, int r, int c) /* The matrix */
{
    for (i=0; i<r; i++)
        free((char *) m[i]);
}

```


【図197】

```

starting = t_cursor + 1; // Initialize cursor
startingC = t_cursorC + 0; // Facing left
t_cursor = LEFT; // Set cursor direction
t_step = right_boundary; // Set step to 1 (outlined)
not_finished = 1; // While we haven't returned to starting
while(not_finished) // Adjust curves for this new position
{
    if (t_cursor < curvel(t_cursor)) // If turtle is left of current break
    {
        curvel(t_cursor) = t_cursor; // Move break
        if (t_cursor < left) // Update left
        {
            left = t_cursor; // Update top and bot
            top = t_cursor; // Update top and bot
            bot = t_cursor;
        }
        if (t_cursor > curvel(t_cursor+1)) // (Same for right also)
        {
            curvel(t_cursor+1) = t_cursor+1; // Note: we've returned to the starting point
            if (t_cursor > right) // and max because the left
            {
                right = t_cursor; // side (above) will get it
            }
        }
        // Check to see if we've entirely circled the image
        if (t_cursor == starting) && (t_cursor == startingC)
        {
            // We've returned to the starting point
            t_cursor = LEFT; // Remember direction
            t_cursorC = 0; // Check the space above
            if ((t_in(t_cursor) & 7) != 1) // If space above is on, and hasn't
            {
                t_cursor = LEFT; // been outlined, don't exit
            }
            if (t_in(t_cursor) & 7) != 1 // (Otherwise for the left space)
            {
                not_finished = 0; // We're finished, exit the while routine
            }
            break;
        }
        t_cursor = 1; // Restore direction and continue
        // Move mouse to next position along the edge
        t_in(t_cursor); // Check area to left of mouse
        if (t_in(t_cursor) & 3) == 1 // If something there
        {
            t_move_forward(); // Move to left
            continue; // And loop again
        }
        t_in(t_cursor); // Check area in front of mouse
        if (t_in(t_cursor) & 5) == 1 // If something there
        {
            t_move_forward(); // Move forward
            continue; // And loop again
        }
        t_in(t_cursor); // Check area to right of mouse
        if (t_in(t_cursor) & 5) == 1 // If something there
        {
            t_move_forward(); // Move to right
            continue; // And loop again
        }
}

```

【図198】

```

t_turn_right(); // Check area to behind of mouse
if ((t_line.front(t_line) & 5) == 1) // If something there
{
    t_move_forward(); // Move behind mouse
    continue; // And loop again
}
not_finished = 0; // Else it's a one-pixel image. so exit
line.pixel(t_cursor)(t_cursor) = line.pixel(t_cursor)(t_cursor) + 1;
// Make this spot

// Load the info into the Charlist
new_char->stop = stop;
new_char->bot = bot; // (Make it exclusive)
new_char->left = left;
new_char->right = right; // (Make it exclusive)
new_char->type = CHARACTER_FULL_OUTLINED;

Charlist_place_new_char(cclist); // Insert the new character

// Mark the character as removed from the list
for (r=0; r<line.size; r++)
for (c=cursorx(t); c<cursorx(t)+1; c++)
    line.pixel[r][c] = line.pixel[r][c] + 1;

////////////////////////////////////
// Main routine to call to add break to the image
////////////////////////////////////
void do_outlining( // Returns number of characters added
    cimage* line, // image of the line
    struct Charlist* cclist, // The character list
    int start_col, // The range to search through (inclusive)
    int stop_col) // (exclusive)
{
    int x, c; // The row and column we're on

    for (c=start_col; c<stop_col; c++) // Search through column range
    for (r=line.size-1; r>=0; r--) // Scan from bottom to top
    {
        if ((line.pixel[r][c] & 5) == 1) // On pixel that hasn't been outlined
        {
            line.pixel[r][c] = 12;
            print_outlining(line, c);
            extract_outline_char(line, cclist, start_col, stop_col, x, c);
            // Outline the character
        }
    }
}

```

【図208】

```

//
// Filename: test_mult.C
// This file is used to test and develop multilevel2.C
//
#include <stdio.h>
#include <string.h>
#include "image.h"
#include "ymalloc.h"
#include "xs.h"
#include "charlist.h"

extern void l4_initialize();
void level4(cimage* line, struct Charlist* cclist);
void xs_init();

{
    struct Charlist cclist;
    struct Character* ch;
    cimage image;
    int i, j;

    l4_initialize();
    Charlist_initialize(&cclist, 100, 100); // 10 chars, 100-max height

    char filename1(100);
    char filename2(100);
    printf("Input filename? /d3/x11/");
    scanf("%s", filename1);
    strcpy(filename2, "/d3/c11/");
    strcat(filename2, filename1);
    image.read_if(filename2);
    image.slipspace();

    ch = &cclist.ch[cclist.size]; // Set new_char to the last character

    // Load the character
    ch->stop = 0;
    ch->bot = image.size;
    ch->left = 0;
    ch->right = image.size;
    for (i=ch->top; i<ch->bot; i++) {
        ch->cursorx[i] = ch->left;
        ch->curvey[i] = ch->right;
    }

    Charlist_place_new_char(&cclist);
    level4(image, &cclist);
    printf("**** End of Program ****\n");
}

```

[図199]

```

//***** This routine will plot a scaled image on the screen *****
void clear_screen_and_plot_image() // The image
{
    float scale;
    int nx, ny, tx, ty;
    int i, j, ix, iy, ix2, iy2;

    // Compute scale to fit image on screen (x and y)
    scale = (float) (WINDOW_SIZE_X-10) / (float) image.sizeX;
    scale = (float) (WINDOW_SIZE_Y-10) / (float) image.sizeY;
    if (scale < scale) // Take the least of the two
        scale = scale; // Only shrink the image
    scale = 1.0;

    // Draw the image
    for (i=0; i<ny; i++)
    {
        for (j=0; j<nx; j++)
        {
            tx = (int) ((double) j / scale);
            ty = (int) ((double) i / scale);
            if (image.pixelP[tx] != 0)
                nx_plotxy(tx, ty, 1);
        }
    }
}

// Draw Box Around Image
int nx, ny;
int ix, iy, ix2, iy2;
for (i=0; i<ny; i++)
    for (j=0; j<nx; j++)
        nx_plotxy(j, i, 1);

//***** This routine will create a histogram from cols m to n.
// THIS WILL INSERT A 1 FOR TEXT 0 FOR BLANK
void make_histogram()
{
    int n; // The input image
    int m; // The histogram stepping col (inclusive)
    int n; // The histogram stepping col (exclusive)
    int n; // The output histogram (allocated previously)
    int n;
}

```

```

// array of histograms; from [row]
int *histogram;
// array of starting test positions
int *startpos;
// array of stopping positions
int *stoppos;
// The cursor for the start and stop tests
int "cursor";
// The site of each list
int *list;
// Include this request in the next line (0=no, 1=yes)
int "include";
// Stop at this line?
int "stop_at_line";
// The division with the topmost test
int rangel;
// The test range in the adjacent division (row 1's)
int rangel;
// The test range in the current division (row i's)
int rangel;
// The row and column of line in page
int r1, c1, c2;
// The minimum line height
int minheight;
// Line rejection flag
int reject;
// New handy variables
int j,k,l,m,n,p,q;

stopcol = {vector(INDIVISIONS);
stopcol = {vector(INDIVISIONS);
begin = {matrix(INDIVISIONS, PAGE_SIZE);
stopcol = {matrix(INDIVISIONS, PAGE_SIZE); // This is overkill;
cursor = {vector(INDIVISIONS);
size = {vector(INDIVISIONS);
include = {vector(INDIVISIONS);
}

min_line_height = {list; // (float; page dot - PSEUDO_2_XIN_BLOCK_SIZE);

////////// Step 1: Define columns and find histograms //////////
// Find range of columns
started[0] = MAXVAL;
for (l=0; l<(INDIVISIONS-1); l++)
{
c = PAGE_COL * MPAGE_COL + (l+1); // INDIVISIONS;
startcol[l+1] = c - OVERLAY;
stopcol[l+1] = c + OVERLAY;
}
stopcol[INDIVISIONS-1] = PAGE_COL;
// Find histogram
for (l=0; l<(INDIVISIONS; l++)
{
matrix(histopage, startcol[l], stopcol[l], Ngranall);
}

////////// Step 1: Make test area lists //////////
for (l=0; l<(INDIVISIONS; l++)
{
// Now, tests are guaranteed on both ends of the histogram
size[l] = 0; // Initialize
for (j=PAGE_ROW+1; j<PAGE_ROW+j; j++)
{
if (Histogram[j][l] == 0) && !granval(j)(0-1)) == 0) // If starting test
startcol[l+1] = j-1;
if (Histogram[j][l] != 0) && !granval(j)(0-1)) == 0) // If stopping test
stopcol[l+1] = j-1;
else(l) += 1;
}
}

////////// Step 1: Find the Lines //////////
for (l=0; l<(INDIVISIONS; l++)
{
current[l] = 0; // Set all pointers to the top of the list
end_of_list = 0;
while(end_of_list == 0)

```

—194—

—196—

[illegible]

—198—

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

【図206】

```

/* This program will test the charlist */
#include "charlist.h"

void print_charlist(clist)
struct Charlist *clist;
{
    struct Character *ch;
    printf("Charlist:\n");
    ch = clist->first;
    while(ch != 0) {
        printf("Loc: %d; Chars: %s\n", ch->loc, ch->chars);
        if (ch->prev == 0)
            printf("****\n");
        else
            printf("%d", ch->prev->stop);
        printf("%d", ch->stop);
        if (ch->next == 0)
            printf("****\n");
        else
            printf("%d", ch->next->stop);
        printf("\n");
        ch = ch->next;
    }
    printf("End of charlist\n\n");
}

main()
{
    struct Character *ch;
    struct Charlist clist;
    int A[10]; /* Dummy vector for height */
    int i, p;
    Charlist_initialize(&clist, 10, 10);
    for (i=0; i<10; i++)
    {
        printf("Position of new char %d: ", i);
        scanf("%d", &p);
        ch = A[clist->cur[clist->size]];
        ch->top = 1;
        ch->bot = i-1;
        ch->curval[i] = ch->curval2[i] = p;
        Charlist_place_new_char(&clist);
        print_charlist(&clist);
    }
    while(1)
    {
        printf("Replace which char? ");
        scanf("%d", &i);
        ch = clist->first;
        while(ch != 0) {
            if (ch->top == i)
                break;
            ch = ch->next;
        }
        if (ch == 0) {
            printf("Character not found.\n");
        } else {
            printf("New position? ");
        }
    }
}

```

```

scanf("%d", &p);
ch->curval[i] = ch->curval2[i] = p;
Charlist_place_new_char(&clist, ch);
print_charlist(&clist);
}

```

【図209】

```

/* This file prints a character to be recognised on the screen by */
/* calling the routine thieu_recognize() */

void thieu_recognize(size_r, size_c, image)
int size_r; /* The number of rows */
int size_c; /* The number of columns */
char **image; /* The image: bit 1 is the image */
{
    int r, c;
    printf("\nThe character is:\n");
    for (r=0; r<size_r; r++)
    {
        for (c=0; c<size_c; c++)
            if (image[r][c] & 1)
                printf(" "); /* On */
            else
                printf("."); /* Off */
        printf("\n");
    }
    printf("***** End of character *****\n\n");
}

```

【図207】

```

// Filename: test_mult.c
// This file is used to test and develop multiwell.c
// =====
#include <stdio.h>
#include "cimages.h"
#include "xmalloc.h"
#include "xs.h"
#include "charlist.h"

extern void xs_initialize();
void level(cimages *img, struct Charlist *clist);

void xs_init()
{
    struct Charlist clist;
    struct Character *ch;
    cimages image;
    int i,j;

    xs_initialize();

    Charlist_initialize(&clist, 10, 100); // 10 chars, 100-max height
    ch = &clist.chr[clist.size]; // Set new_char to the last character

    // =====
    // Load the image
    //
    image.fill(1,10,10);
    image.pixel[6][6] = 0;
    image.pixel[7][6] = 0;
    image.pixel[8][6] = 0;
    image.pixel[5][7] = 0;
    image.pixel[7][7] = 0;
    image.pixel[8][7] = 0;
    image.pixel[6][8] = 0;
    image.pixel[7][8] = 0;
    image.pixel[8][8] = 0;
    image.pixel[6][9] = 0;
    image.pixel[7][9] = 0;
    image.pixel[8][9] = 0;
    image.pixel[2][10] = 0;
    image.pixel[3][10] = 0;
    image.pixel[4][10] = 0;
    image.pixel[5][10] = 0;
    image.pixel[6][10] = 0;
    image.pixel[7][10] = 0;
    image.pixel[8][10] = 0;
    image.pixel[2][11] = 0;
    image.pixel[3][11] = 0;
    image.pixel[4][11] = 0;
    image.pixel[5][11] = 0;
    image.pixel[6][11] = 0;
    image.pixel[7][11] = 0;
    image.pixel[8][11] = 0;
    image.pixel[0][12] = 0;
    image.pixel[1][12] = 0;
    image.pixel[2][12] = 0;
    image.pixel[3][12] = 0;
    image.pixel[0][13] = 0;
    image.pixel[1][13] = 0;
    image.pixel[2][13] = 0;
    image.pixel[3][13] = 0;

    // Load the character
    ch->top = 0;
    ch->bot = 10;
    ch->left = 0;
    ch->right = 35;
    for (i=ch->top; i<ch->bot; i++) {
        ch->curve1[i] = ch->left;
        ch->curve2[i] = ch->right;
    }

    Charlist_place_new_char(&clist);
    level(image, &clist);
    printf("**** End of Program ****\n");
}

```

【図214】

```

/* If your routines are in C++ then CPP must be defined in make file */
#ifdef CPP
extern "C" void xs_set_foreground(int x, int y, int b);
extern "C" void xs_set_background(int x, int y, int b);
extern "C" void xs_init();
extern "C" void xs_flush();
extern "C" void xs_delay(int x, int y);
extern "C" void xs_unplotq(int x, int y);
extern "C" void xs_redisplay();
extern "C" void xs_get_image(int x, int y, int width, int height, char *buf);
extern "C" void xs_get_win();
extern "C" void xs_draw_box(int x, int y, int width, int height);
extern "C" void xs_string(int x, int y, char *str);
extern "C" void xs_setcolor(int color);
extern "C" void xs_vla_dim(int *width, int *height);
#endif

```

【图210】

```

unsigned char *; /* The character to slice bit(s) from */
int b; /* The beginning of the bit number */
int a; /* Number of bits to get */
{
    return(x >> (a-b-1)) & ~(~0 << a);
}

/*
Routine: uc_ascii_bitmapwidth,height,buf()
Description: Display the bitmap using ascii characters
Returns: Nothing
Author: Rick Lee
*/
uc_ascii_bitmapwidth,height,buf()
{
    int width;
    int height;
    char *buf;

    short bit_code = 0x0000;
    short bit_result = 0;
    short x,y;
    for(x = 0; x < height; x++)
    {
        for(y = 0; y < width; y++)
        {
            bit = buf[(x+y)];
            bit_result = bit & bit_code;
            if(bit_result == 0)
                printf(" ");
            else
                printf("X");
            bit_code = bit_code >> 1;
        }
        bit_code = 0x0000;
        printf("\n");
    }
}

/*
Routine: uc_reversewidth,height,buf()
Description: Reverse the bits for display format
Returns: Nothing
Author: Rick Lee
*/
uc_reversewidth,height,buf()
{
    int width;
    int height;
    char *buf;

    unsigned char val,new_val;
    int x,y;
    for(x = 0; x < (width-1); height; x++)
    {
        x = 0;
        val = buf[x];
        new_val = 0;
        for(y = 0; y < height; y++)
        {
            if(uc_getbits(val,x,y))
                new_val = (new_val << 1) | new_val;
        }
        buf[x] = new_val;
    }
}

/*
Routine: uc_getbits(x,y,n)
Description: Check if the bit is turned on or not
Returns: Nothing
Author: Rick Lee
*/
uc_getbits(x,y,n)

```

—202—

[illegible]

【図212】

```

/* Routine: aa_redisplay()
Description: Redisplay the "stuff" that was on the window
Return: Nothing
Author: Rick Lee
*/
aa_redisplay()
{
    XClearArea(XDisplay(canvas), XWindow(canvas), 0, 0, 0, TRUE);

    /* Routine: aa_draw_box(x, y, width, height)
Description: Draw a box on screen
Return: Nothing
Author: Rick Lee
*/
aa_draw_box(x, y, width, height)
{
    int x;
    int y;
    int width;
    int height;
    {
        XDrawRect(canvas, XWindow(canvas), line_gc,
            x, y, width, height);
    }

    /* Routine: aa_string
Description: Draw a string on the screen
Return: Nothing
Author: Rick Lee
*/
aa_string(x, y, str)
{
    XDrawText(canvas, XWindow(canvas), text_gc, line_gc,
        x, y, str, strlen(str));

    /* Routine: aa_setcolor
Description: Set the current color
Return: Nothing
Author: Rick Lee
*/
aa_setcolor(color)
{
    int color;
    {
        gc = XGCValues[0];
        XSetForeground(XDisplay(canvas), line_gc, color);
    }

    /* Routine: aa_cle_wm()

```

【図213】

```

/*
Routine: xs_win_d1
Description: Get the size of the window
Returns: Nothing
Author: Rich Lee
*/
xs_win_d1(width,height)
int *width;
int *height;
{
    Argv argv[2];
    Dimension win_width,win_height;

    XSetArg(argv[0],XtWidth,win_width);
    XSetArg(argv[1],XtHeight,win_height);
    XGetValues(canvas,argv,2);

    *width = win_width;
    *height = win_height;
}

/*
xs_get_cmap(w)
Widget w;
{
    int ncolors;
    XColor colors[256];
    Colormap my_cmap;
    int count = 1;

    Display *d;
    int scr;
    Colormap def_colormap;
    XColor color;
    int x;
    int the_color = 2000;

    d = XDisplay(w);
    scr = DefaultScreen(d);
    ncolors = DisplayCells(d,scr);
    def_colormap = DefaultColormap(d,scr);
    for(x = 0; x < ncolors; x++)
    {
        colors[x].pixel = x;
        colors[x].flags = DoRed | DoGreen | DoBlue;

        colors[x].red = the_color;
        colors[x].green = the_color;
        colors[x].blue = the_color;
        the_color -= 100;
    }
    XQueryColors(d,def_colormap,colors,ncolors);

    my_cmap = XCreateColormap(d,DefaultRootWindow(d),DefaultVisual(d,scr),
        AllocAll);
    XStoreColors(d,my_cmap,colors,ncolors);
}

```

【図228】

```

c1 = get2bytes(f, fp);
c2 = get2bytes(f, fp);
return c1 * c2*65536;

////////////////////////////////////
// Plot image using Flash Graphics at (xloc, yloc) with color
void elimage(plot(int x, int y)
{
    int px,py;          // Pixel coords on the screen (upper left)
    char color;

    for (py=0; py<ysize; py++)
        for (px=0; px<xsize; px++)
        {
            color = pixel[py][px];
            if (color & 1)
                xs_plotxy(px,py); // Draw it
        }
}

////////////////////////////////////
// Plot image with a box around it (with color 1)
void elimage(plotbox(int x, int y)
{
    int px,py;

    for (px=0; px<xsize; px++) {          // Draw horizontal lines
        if ((x-px-y)%2 == 0)
            xs_plotxy(x-px, y);
        if ((x+px-y-sizeX)%2 == 0)
            xs_plotxy(x+px, y-sizeY);
    }
    for (py=1; py<ysize-1; py++) {
        if ((x-y-py)%2 == 0)
            xs_plotxy(x, y-py);
        if ((x-sizeX+py-y)%2 == 0)
            xs_plotxy(x-sizeX-1, y-py);
    }
    plot(x-1, y-1);          // Plot the image
}

```

【図240】

```

/* This file prints a character to be recognized on the screen by */
/* calling the routine thieu_recognize() */

char thieu_recognize(size_r, size_c, image)
int size_r;          /* The number of rows */
int size_c;          /* The number of columns */
char **image;        /* The image: Bit 1 is the image */

{
    int r,c;
    char input_string[100];

    printf("\nThe character is:\n");
    for (r=0; r<size_r; r++)
    {
        for (c=0; c<size_c; c++)
            if (image[r][c] & 1)
                printf(" ");          /* On */
            else
                printf(".");          /* Off */
        printf("\n");
    }
    printf("***** End of character *****\n\n");
    printf("Enter Character? ");
    scanf("%s", input_string);
    return input_string[strlen(input_string)-1];
}

```

—205—

[illegible]

【図216】

```

// Filename: zerovvp.c
// This routine contains the lines to find blocks that have non-zero vvp
// (level 1 segmentation)
//
// =====
//
#include "charlist.h"
#include "images.h"
// =====
// Scan a line quickly (every 1/4 pixel) looking for an 'on' pixel
int line_scan_quick( // Returns 1 for pixel found, 0 otherwise
    cimages line, // The line to process
    int c) // The column to scan
{
    int r;

    for (r=0; r<line.sizeh; r++)
        if ((line.pixel[r][c] & 1))
            return 1;
    return 0;
}

// =====
// Scan a line completely looking for an 'on' pixel
int line_scan_complete( // Returns 1 for pixel found, 0 for otherwise
    cimages line, // The line
    int c) // The column to scan
{
    int r;

    for (r=0; r<line.sizeh; r++)
        if ((line.pixel[r][c] & 1))
            return 1;
    return 0;
}

// =====
// The main routine. Call this to make the list
void extract_zero_vvp_list(
    cimages line, // The image of the line to segment
    struct NonZeroVvpList *list) // A pointer to the list
{
    int c;
    int left, right;

    for (c=0; c<line.sizec; c++) // Scan the image
    {
        if (line_scan_quick(line, c) // If found something
        {
            left = right = c; // Look to the left for the start
            while (--c >= 0)
                if (line_scan_complete(line, c) == 0)
                    break;
            left = c+1;
            c = right; // Keep looking to the right
            while (++c < line.sizec)
                if (line_scan_complete(line, c) == 0)
                    break;
            right = c;
            zerovvp_add(list, left, right); // Add in the new break
        }
    }
}

```

【図245】

```

/* If your routines are in C++, then CPP must be defined in make file */
#ifdef CPP
extern "C" void xs_init();
extern "C" void xs_flush();
extern "C" void xs_plotxy(int x, int y);
extern "C" void xs_unplotxy(int x, int y);
extern "C" void xs_redisplay();
extern "C" void xs_dis_bitmap(int x, int y, int width, int height, char *buf);
extern "C" void xs_clr_win();
extern "C" void xs_draw_box(int x, int y, int width, int height);
extern "C" void xs_string(int x, int y, char *str);
extern "C" void xs_setcolor(int color);
extern "C" void xs_win_dim(int *width, int *height);
#endif

```


【図219】

```

/* filename: autolinput.h
 * This file contains the code to input from a filename
 * .....
#include <stdio.h>
#include "autolinput.h"

/* Global variables */
int auto_eof_detected;
FILE *auto_fp;
int save_to_file;
int save_count;
char filename[100];

void auto_initialize()
{
    int not_exit = 1;
    auto_eof_detected = 0;
    while(not_exit)
    {
        printf("Input filename for autolinput? ");
        scanf("%s", filename);
        auto_fp = fopen(filename, "r");
        if (auto_fp == NULL) /* lead open failed */
        {
            auto_fp = fopen(filename, "w");
            if (auto_fp == NULL) /* twice open failed */
            {
                printf("cannot open file '%s'\n", filename);
            }
            else /* file opened as a new opened new file */
            {
                printf("Input 1 at the next prompt.\n");
                save_to_file = 1;
                auto_eof_detected = 1;
                not_exit = 0;
            }
        }
        else /* lead open succeeded */
        {
            printf("Opening existing file '%s'.\n", filename);
            not_exit = 0;
        }
    }

    printf("Do you wish to save the auto-input into this file? (1 = yes) ");
    void auto_terminate()
    {
        FILE *file = autolinput_int();
        printf("Warning! Change the file to a 'p' to turn off appending to the file!\n");
        fclose(auto_fp);
        save_count = 0;
    }

    int auto_input_int()
    {
        int i;
        if (auto_eof_detected)
            printf("\nInput input [%d = newline, %999 = exit] ", i);
    }
}

```

```

scanf("%d", &i);
if (i == -9999)
{
    fclose(auto_fp);
    exit(1);
}
if (save_to_file == 1)
{
    if (i == -9) /* Print new lines */
    {
        fprintf(auto_fp, "\n"); /* Print the new lines */
        fflush(auto_fp);
        save_count = 0;
        return auto_input_int(); /* And input again */
    }
    printf(auto_fp, "%d ", i);
    fflush(auto_fp);
    if (++save_count > 10)
    {
        save_count = 0;
        printf(auto_fp, "\n");
    }
    return i;
}
if (fclose(auto_fp, "d", &i) == EOF)
{
    printf("Warning! EOF OF FILE DETECTED ----\n");
    if (save_to_file == 1)
    {
        fclose(auto_fp);
        auto_fp = fopen(filename, "a");
        if (auto_fp == NULL)
        {
            printf("Warning! WARNING! CANNOT APPEND TO FILE. Is '%s'\n", filename);
            save_to_file = 0;
        }
        fprintf(auto_fp, "\n");
    }
    auto_eof_detected = 1;
    return auto_input_int();
}
printf("%d\n", i);
return i;
}

```

【图 2 2 1】

—209—

[図222]

```

/* This routine will insert the character beyond the charlist (charlist),
 * loc is recalculated for each character and the new character, however
 * the new character is not added to the list until after the character,
 * extremely good idea to resort the charlist!
 */
void Charlist_Insert_After_Charlist(charlist, ch)
struct Charlist *charlist; /* Place the new character after this one */
struct Character *newchar; /* Pointer to the new character adding */
{
    struct Character *newchar; /* Set newchar */
    /* Calculate the position of the new character */
    Charlist_Calculate_Loc(charlist, loc);
    /* Check for full list */
    if (charlist->next == charlist->first) /* If too big... */
        return; /* Increment the size of the list */
    /* Error in Charlist_Calculate_Loc */
    printf("Error in Charlist_Calculate_Loc: Character list is FULL\n");
    exit(1);
}

/* Remove the character from the list */
void Charlist_Remove_Charlist(charlist, ch)
struct Character *newchar; /* Set newchar */
{
    /* Calculate the position of the new character */
    Charlist_Calculate_Loc(charlist, loc);
    /* Check for full list */
    if (charlist->next == charlist->first) /* If too big... */
        return; /* Increment the size of the list */
    /* Error in Charlist_Calculate_Loc */
    printf("Error in Charlist_Calculate_Loc: Character list is FULL\n");
    exit(1);
}

/* This routine will remove a character from the character list
 * Note: The memory is not reclaimed!! So don't rely on this too much.
 * Also, prev and next pointers of ch are not altered!
 */
void Charlist_Remove_Charlist(charlist, ch)
struct Character *newchar; /* Set newchar */
{
    /* Calculate the position of the new character */
    Charlist_Calculate_Loc(charlist, loc);
    /* Check for full list */
    if (charlist->next == charlist->first) /* If too big... */
        return; /* Increment the size of the list */
    /* Error in Charlist_Calculate_Loc */
    printf("Error in Charlist_Calculate_Loc: Character list is FULL\n");
    exit(1);
}

/* This routine will place a character (that is not in the charlist)
 * into the charlist.
 */
void Charlist_Place_Charlist(charlist, ch)
struct Character *newchar; /* Set newchar */
{
    /* Calculate the position of the new character */
    Charlist_Calculate_Loc(charlist, loc);
    /* Check for full list */
    if (charlist->next == charlist->first) /* If too big... */
        return; /* Increment the size of the list */
    /* Error in Charlist_Calculate_Loc */
    printf("Error in Charlist_Calculate_Loc: Character list is FULL\n");
    exit(1);
}

/* This routine will place a character (that is not in the charlist)
 * into the charlist.
 */
void Charlist_Place_Charlist(charlist, ch)
struct Character *newchar; /* Set newchar */
{
    /* Calculate the position of the new character */
    Charlist_Calculate_Loc(charlist, loc);
    /* Check for full list */
    if (charlist->next == charlist->first) /* If too big... */
        return; /* Increment the size of the list */
    /* Error in Charlist_Calculate_Loc */
    printf("Error in Charlist_Calculate_Loc: Character list is FULL\n");
    exit(1);
}

```

【図 2 2 4】

[illegible]

—213—

```

for (i=0; i<C1; i++)
    temp_pixel[i*(i-C1)+C1] = pixel[i](i);
}

allocate(temp_align, temp_sizeC); // Copy temp image to current image
for (i=0; i<temp_sizeC; i++)
    for (j=0; j<temp_sizeC; j++)
        pixel[i](j) = temp_pixel[i](j);
}

// Read in part of a TIFF file
void climage::readtiffchar (filename)
{
    FILE *tagfile; // Input file
    long int tagfile_position; // the our file position (in bytes)
    long int width; // Width of picture
    long int height; // Height of picture
    long int black; // Value of black pixel
    long int white_offset; // Offset per strip
    long int numstrips; // Number of strips
    long int numtags; // tag, type, len, value // Tag values
    long int i;

    if (tagfile = fopen(filename, "rb")) == NULL
    {
        cout << "Error in opening " << filename << "\n";
        exit(1);
    }

    tagfile_position = 0;

    // Read in header and make sure it's small
    i = getbytes(tagfile, tagfile_position);
    if (i != 0x00000000)
    {
        cout << "... Error: TIFF file not in Intel format\n";
        exit(1);
    }

    // Read in Magic number
    i = getbytes(tagfile, tagfile_position);
    if (i != 0x00000000)
    {
        cout << "... BAD Magic Number for TIFF file\n";
        exit(1);
    }

    // Find first file directory
    i = getbytes(tagfile, tagfile_position);
    while (tagfile_position < i)
    {
        i = getbytes(tagfile, tagfile_position);
    }

    // Tags in Directory
    numtags = getbytes(tagfile, tagfile_position);
    while (--numtags >= 0)
    {
        tag = getbytes(tagfile, tagfile_position);
        type = getbytes(tagfile, tagfile_position);
        len = getbytes(tagfile, tagfile_position);
        value = getbytes(tagfile, tagfile_position);
        if ((len & 1) && (tag & 32768) == 0)
        {
            cout << "Error: Tag length is 1\n";
            exit(1);
        }
    }
}

// With the new image (size size adjusted);
allocate(R1, C1);
for (int i=0; i<sizeC; i++)
    pixel[i](i) = value;
}

// Copy a section of a image into another. image = reference image
// Range is (R1, R2) (C1, C2)
void climage::clipimage (image, int R1, int R2, int C1, int C2)
{
    int i, j, k;

    // Swap R1 & R2
    i = R1; R1 = R2; R2 = i;

    // Swap C1 & C2
    i = C1; C1 = C2; C2 = i;

    // Check limits
    if (R1 < 0) R1 = 0;
    if (R2 < 0) R2 = 0;
    if (C1 < 0) C1 = 0;
    if (C2 < 0) C2 = 0;
    if (R1 > image.height) R2 = image.height;
    if (R2 > image.height) R1 = image.height;
    if (C1 > image.width) C2 = image.width;
    if (C2 > image.width) C1 = image.width;

    allocate(R2-R1, C2-C1);
    dual = image.dual;
    for (i=R1; i<R2; i++)
        for (j=C1; j<C2; j++)
            pixel[i](j) = image.pixel[i-R1](j-C1);
}

// Clip image, remove the size of the image by moving the dual
// space around it. This adjusts the image size.
void climage::cliprect()
{
    int R1, R2, C1, C2; // The starting and stopping rows of new image
    struct climage temp; // Temporary image

    R1 = sizeR; // Initialize to extreme
    R2 = 0;
    C1 = sizeC;
    C2 = 0;

    // Compute dimensions of new image
    for (i=0; i<sizeC; i++)
        for (j=0; j<sizeR; j++)
        {
            if (pixel[i](j) != 0)
            {
                if (i < R1) R1 = i;
                if (i > R2) R2 = i;
                if (j < C1) C1 = j;
                if (j > C2) C2 = j;
            }
        }

    if (R1 > R2) R1 = R2 = 0;
    if (C1 > C2) C1 = C2 = 0;

    temp.allocate(R2-R1, C2-C1);
    for (i=R1; i<R2; i++)
        for (j=C1; j<C2; j++)
            temp.pixel[i-R1](j-C1) = pixel[i](j);
}

```


【図227】

```

while (tagfile_position < raster_offset)
{
    i = getByte(tagfile, tagfile_position);
}
//----- READ IN FILE -----
allocated(int) length, (int) width;
int vi;
for (i = 0, i < length; i++)
{
    for (j = 0; j < width; j++)
    {
        if (i % 4 == 0)
        {
            v = get(tagfile);
            tagfile_position++;
            if (v == EOF)
            {
                cerr << "Error: Premature End of file found in tiff file\n";
                exit(1);
            }
            if (i % 4 != 0)
            {
                if (i % 4 == 1)
                {
                    pixel[i][j] = 0;
                }
                else
                {
                    pixel[i][j] = 1;
                }
            }
            v = v << 3;
        }
    }
}
fclose(tagfile);
//----- READ IN FILE -----
// Read in a byte. File position is updated
// eoferror = 1 to error on EOF, 0 not
int cimage, getByte(tiff *r, long into file_position, int eoferror);
int ci;
if (ci == EOF)
{
    cerr << "Error: Premature End of file (in getByte)\n";
    exit(1);
}
file_position++;
return ci;
//----- READ IN FILE -----
// Read in an integer from file (int, long)
long int cimage, getByte(tiff *r, long into fp);
int ci, ci2;
ci = getByte(r, fp, 1);
ci2 = getByte(r, fp, 1);
return ci << ci2;
//----- READ IN FILE -----
// Read in a long integer
long int cimage, getByte(tiff *r, long into fp);
long int ci, ci2;

```

```

switch (tag)
{
    case 0x0001: // NewSubFileType
    {
        if (value != 0)
        {
            cerr << "Can't deal with non-standard NewSubFileType\n";
            exit(1);
        }
        break;
    }
    case 0x0002: // NewImageType
    {
        if (value != 1)
        {
            cerr << "Offset type not full resolution\n";
            exit(1);
        }
        break;
    }
    case 0x0003: // Width
    {
        width = value;
        break;
    }
    case 0x0004: // Length
    {
        length = value;
        break;
    }
    case 0x0005: // Bits per sample
    {
        if (value != 1)
        {
            cerr << "Error: Bits/sample must be 1\n";
            exit(1);
        }
        break;
    }
    case 0x0006: // Compression
    {
        if (value != 1)
        {
            cerr << "Compression: bad Error\n";
            exit(1);
        }
        break;
    }
    case 0x0007: // Black is
    {
        black = value;
        break;
    }
    case 0x0008: // Thresholding tag
    {
        if (value < 1 || value > 3)
        {
            cerr << "Thresholding tag = illegal value\n";
            exit(1);
        }
        break;
    }
    case 0x0009: // Offset to raster data
    {
        raster_offset = value;
        break;
    }
    case 0x0010: // bps = value
    {
        bps = value;
        break;
    }
    case 0x0011: // case 0x0011b
    {
        break;
    }
    case 0x0012: // case 0x0012b
    {
        break;
    }
    default:
    {
        if (value < 0 || value > 0)
        {
            cerr << "Warning: An unsupported tag appears in the tiff file\n";
        }
        break;
    }
}
// while

```

【図229】

```

// Filename: cimages.h
// This is the header file for cimages.C
//
// =====
// Define XEM_DEBUG
// =====
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

class cimage; // Forward reference

// =====
// CLASS: cimage
// =====
class cimage
{
public:
    int sizeR; // Number of rows (first index of array)
    int sizeC; // Number of cols. (second index of array)
    int allocated_sizeR; // Amount allocated
    int allocated_sizeC; // Amount allocated
    int dpi; // Dots per inch of the image (0 if unknown)
    unsigned char **pixel; // The binary image (2d array)

    cimage(); // Initialize
    ~cimage(); // Destroy
    cimage(int r, int c); // Initialize with a size
    cimage(cimage& temp);

    void reallocate(int R, int C); // Will allocate memory for an image
    void fill(char value, int R, int C); // Fill image with a constant value
    void clip(cimage& image, int R1, int R2, int C1, int C2); // Clip image
    void clipimage(); // Clip zero space surrounding image
    void readtiff(char *filename); // Read in a tiff image
    void plot(int x, int y); // Plot binary image
    void plotbox(int x, int y); // Plot with a box around it

private:
    void allocate(int r, int c); // Allocate memory
    void deallocate(); // Deallocate memory
    int getbyte(FILE *f, long int& file_position, int eoferror);
    long int getbytes(FILE *f, long int& fp);
    long int getbytes(FILE *f, long int& fp);
};

```


【圖 2 3 2】

[illegible]

【図233】

```

/* This program contains common vector and matrix routines */
#include <stdio.h>
#include <stdlib.h>

/* Routine for error during memory allocation */
void allocation_error()
{
    char *s;
    /* The error text */
    printf("Memory allocation error %s\n", s);
    fprintf(stderr, "Error: %s\n", s);
    exit(1);
}

/* Allocate an integer vector of size [0..n-1] */
int *vector(int n)
{
    int *v;
    /* Size of vector */
    v = (int *) malloc(sizeof(int) * n);
    if (!v) allocation_error("in vector");
    return v;
}

/* Free an integer vector v of size [0..n-1] */
void free_vector(int *v, int n)
{
    free((int *) v);
}

/* Allocate an integer matrix of size [0..r-1][0..c-1] */
int **matrix(int r, int c)
{
    int **m;
    /* The size of the matrix */
    m = (int **) malloc(sizeof(int *) * r);
    for (int i = 0; i < r; i++)
        m[i] = (int *) malloc(sizeof(int) * c);
    if (!m || !m[0]) allocation_error("in matrix");
    return m;
}

/* Free an integer matrix m of size [0..r-1][0..c-1] */
void free_matrix(int **m, int r, int c)
{
    for (int i = 0; i < r; i++)
        free((int *) m[i]);
    free((int **) m);
}

/* Allocate an integer matrix of size [0..r-1][0..c-1] */
char **matrix(int r, int c)
{
    char **m;
    /* The size of the matrix */
    m = (char **) malloc(sizeof(char *) * r);
    for (int i = 0; i < r; i++)
        m[i] = (char *) malloc(sizeof(char) * c);
    if (!m || !m[0]) allocation_error("in matrix");
    return m;
}

/* Free an integer matrix m of size [0..r-1][0..c-1] */
void free_matrix(char **m, int r, int c)
{
    for (int i = 0; i < r; i++)
        free((char *) m[i]);
}

/* Allocate an integer vector of size [0..n-1] */
int *vector(int n)
{
    int *v;
    /* Size of vector */
    v = (int *) malloc(sizeof(int) * n);
    if (!v) allocation_error("in vector");
    return v;
}

/* Free an integer vector v of size [0..n-1] */
void free_vector(int *v, int n)
{
    free((int *) v);
}

/* Allocate an integer matrix of size [0..r-1][0..c-1] */
int **matrix(int r, int c)
{
    int **m;
    /* The size of the matrix */
    m = (int **) malloc(sizeof(int *) * r);
    for (int i = 0; i < r; i++)
        m[i] = (int *) malloc(sizeof(int) * c);
    if (!m || !m[0]) allocation_error("in matrix");
    return m;
}

/* Free an integer matrix m of size [0..r-1][0..c-1] */
void free_matrix(int **m, int r, int c)
{
    for (int i = 0; i < r; i++)
        free((int *) m[i]);
    free((int **) m);
}

/* Allocate an integer matrix of size [0..r-1][0..c-1] */
char **matrix(int r, int c)
{
    char **m;
    /* The size of the matrix */
    m = (char **) malloc(sizeof(char *) * r);
    for (int i = 0; i < r; i++)
        m[i] = (char *) malloc(sizeof(char) * c);
    if (!m || !m[0]) allocation_error("in matrix");
    return m;
}

/* Free an integer matrix m of size [0..r-1][0..c-1] */
void free_matrix(char **m, int r, int c)
{
    for (int i = 0; i < r; i++)
        free((char *) m[i]);
}

```

—220—

[illegible]

-221-

[illegible]

【図238】

```

line_fill(0, 255, 0); // Fill up the image with zeros
for (i=0; i<divisions; i++)
{
    if (include[i] == 1)
    {
        for (c=startcol(i); c<stopcol(i); c++)
        {
            line_plot(r-i, (c-cl) * (line_page_pixels/c));
        }
        line_cliparea(); // Remove white space around the line
    }
}

// Step 1a: Do we reject this image? // Default is accept
reject = 0;
if ((r2-cl) < min_line_assign)
{
    reject = 1;
}
printf("image range is %d..%d..%d..%d\n", cl, c2, r1, r2);

// Step 4: Process line // Default is 0
if (reject == 0)
{
    // Clear waste and plot image line;
    printf("Enter 0 to add character segmentation, 1 to ignore");
    if (f == 0)
    {
        process_line(line);
    }
    // Step 1b: Line page, line, elist, r1, c2;
}

// Step 1c: Loop to right and see what column match
// range1 = startcol(i); range2 = stopcol(i);
// if (overlapping, range1, range2, crange) > CLOSE_OVERLAP
// if (cursor(i) < size(i)) // If we're past the end, don't do it
{
    crange1 = startcol(i); crange2 = stopcol(i);
    if (overlapping, range1, range2, crange1, crange2) > CLOSE_OVERLAP
    {
        // Include this column also
        // Adjust range of r1, r2, c1, c2
        if (crange1 < r1)
        {
            r1 = crange1;
        }
        if (crange2 > r2)
        {
            r2 = crange2;
        }
        c1 = startcol(i);
        range1 = crange1;
        range2 = crange2;
    }
}

// Step 1d: Loop to left and see what column match
// range1 = startcol(i); range2 = stopcol(i);
// if (overlapping, range1, range2, crange) > CLOSE_OVERLAP
// if (cursor(i) < size(i)) // If we're past the end, don't do it
{
    crange1 = startcol(i); crange2 = stopcol(i);
    if (overlapping, range1, range2, crange1, crange2) > CLOSE_OVERLAP
    {
        // Include this column also
        // Adjust range of r1, r2, c1, c2
        if (crange1 < r1)
        {
            r1 = crange1;
        }
        if (crange2 > r2)
        {
            r2 = crange2;
        }
        c2 = stopcol(i);
        range1 = crange1;
        range2 = crange2;
    }
}

// Step 1e: Copy the places into the image // Default is 0
// Optimize this. Selectively do fill() and cliparea()

```

【図239】

```

int border_ignore;
int i;

charlist_initialize(&telset, MAX_CHARS_PER_LINE, MAX_LINE_HEIGHT);
process_line.fill(0, 255); // Allocate the memory for the lines

xs_win_dim(&WINDOW_SIZE_X, &WINDOW_SIZE_Y); // Get the window size

printf("Input tiff filename? /d2/cif/");
strcpy(filename1, "/d2/cif/cvsh1.tif");
scanf("%s", filename1);
strcpy(filename, "/d2/cif/");
strcat(filename, filename1);

auto_initialize(); // Initialize Auto-Input routines

printf("Enter courcut average width? ");
COURCUT_AVO_WIDTH = auto_input_int();

printf("Enter courcut delta? ");
COURCUT_DELTA = auto_input_int();

page.dpi = 100;
border_ignore = 1;
// cout << "Input border width to ignore (in pixels)? ";
// cin >> border_ignore;

printf("Reading in File... ");
page.readtiff(filename);
printf("Done.\n");

// setup_init(page); // Initialize page printing routines
// border_ignore = (int) ((float) page.dpi * DPI_3_BORDER_IGNORE);
// printf("border ignore = %d\n", border_ignore);

PAGE_R1 = border_ignore;
PAGE_R2 = page.sizeR - border_ignore; // Define page size
PAGE_R3 = PAGE_R2 - PAGE_R1;
PAGE_C1 = border_ignore;
PAGE_C2 = page.sizeC - border_ignore;
PAGE_C3 = PAGE_C2 - PAGE_C1;

// ----- THIS IS JUST FOR TESTING -----
xs_flush();
printf("Enter 0 to continue? ");
i = auto_input_int();

// clear_scale_and_plot_image(page);

printf("Pausing because page is displayed. Input 0 to segment page? ");
border_ignore = auto_input_int();

while(1) {
    segment_page(page, &elist);
    xs_flush();

    printf("Program finished. Enter 0 to run again? ");
    int itemp;
    itemp = auto_input_int();
}

Charlist_terminate(&telset);
auto_terminate();

```

【図241】

```

/* FILE: util.c
**
** This file contains general utility routines.
**
** ut_reverse(width,height,buf);
** ut_getbits(x,p,n);
** ut_setbits(x,p,n);
** ut_ascii_bitmap(width,height,buf);
**
** Usage:
**
** Just call the routines.
**
** -----Includes-----
** #include <stdio.h>
** #include <stdlib.h>
**
** -----Global-----
**
** -----Externs-----
**
** Routine: ut_reverse(width,height,buf)
** Description: Reverse the bits for display format
** Return: Nothing
** Author: Rick Lee
**
** ut_reverse(width,height,buf)
** int width;
** int height;
** char *buf;
**
** unsigned char val,new_val;
** int x,y;
** for(x = 0; x < (width*2)-height; x++)
** {
**     x = 0;
**     val = buf[x];
**     new_val = 0;
**     for(y = 1; y >= 0 & y-- > 0)
**     {
**         if(ut_getbits(val,x,y))
**             new_val = 0x01 << y | new_val;
**     }
**     buf[x] = new_val;
** }
**
** Routine: ut_getbits(x,p,n)
** Description: Check if the bit is turned on or not
** Return: Nothing
** Author: Rick Lee
**
** ut_getbits(x,p,n)

```

```

unsigned char x; /* The character to place bits in from */
int p; /* The beginning of the bit number */
int n; /* Number of bits to get */
return (x >> (p-n)) & ~((1 << n) - 1);
}

/* Routine: ut_ascii_bitmap(width,height,buf)
** Description: Display the bitmap using ascii characters
** Return: Nothing
** Author: Rick Lee
**
** ut_ascii_bitmap(width,height,buf)
** int width;
** int height;
** char *buf;
**
** short bit_code = 0x0000;
** short bit_result;
** short x,y;
** for(x = 0; x < width; x++)
** {
**     for(y = 0; y < height; y++)
**     {
**         bit_code = buf[x];
**         bit_result = bit_code & 0x01;
**         if(bit_result == 0)
**             printf(" ");
**         else
**             printf("x");
**         bit_code = bit_code >> 1;
**     }
**     bit_code = 0x0000;
**     printf("\n");
** }

```

—225—

[illegible]

【図243】

```

XCopyArea(XtDisplay(canvas), Pix.XtWindow(canvas), line_gc, 0, 0,
          width, height, x, y);

/* Now free the bitmap */
XFreePixmap(XtDisplay(canvas), bitmap);

/* Now free the bitmap */
XFreePixmap(XtDisplay(canvas), pix);

/* Now flush the queue to X-server */
XFlush(XtDisplay(canvas));
}

/*
Routine: xs_clr_win()
Description: Clear the screen
Return: Nothing
Author: Rick Lee
*/
xs_clr_win()
{
    XClearArea(XtDisplay(canvas), XtWindow(canvas), 0, 0, 0, TRUE);
}

/*
Routine: xs_draw_box(x, y, width, height)
Description: Draw a box on screen
Return: Nothing
Author: Rick Lee
*/
xs_draw_box(x, y, width, height)
int x;
int y;
int width;
int height;
{
    XDrawRectangle(XtDisplay(canvas), XtWindow(canvas), line_gc,
                  x, y, width, height);
}

/*
Routine: xs_string
Description: Draw a string on the screen
Return: Nothing
Author: Rick Lee
*/
xs_string(x, y, str)
int x;
int y;
char *str;
{
    XDrawString(XtDisplay(canvas), XtWindow(canvas), line_gc,
                x, y, str, strlen(str));
}

```

```

/*
Routine: xs_setcolor
Description: Set the current color
Return: Nothing
Author: Rick Lee
*/
xs_setcolor(color)
int color;
{
    gov.foreground = color;
    XSetForeground(XtDisplay(canvas), line_gc, color);
}

/*
Routine: xs_win_dim
Description: Get the size of the window
Return: Nothing
Author: Rick Lee
*/
xs_win_dim(width, height)
int *width;
int *height;
{
    Arg args[2];
    Dimension win_width, win_height;

    XSetArg(args[0], XtWidth, win_width);
    XSetArg(args[1], XtHeight, win_height);
    XGetValues(canvas, args, 2);

    *width = win_width;
    *height = win_height;
}

/*
xs_set_map(V)
Widget w;
{
    int ncolors;
    XColor colors[256];
    Colormap my_map;
    int count = 1;

    Display *d;
    int scr;
    Colormap def_colormap;
    XColor Color;
    int w;
    int the_color = 2010;

    d = XtDisplay(w);
    scr = DefaultScreen(d);
    ncolors = DisplayCells(d, scr);
    def_colormap = DefaultColormap(d, scr);
    for(x = 0; x < ncolors; x++)
    {
        colors[x].pixel = ...
    }
}

```

【图 2 4 6】

【図247】

```

// Filtrate: isrovp.c
// This routine contains the lines to find blocks that have non-zero vpp
// (level 1 segmentation)
//
// =====
//
// #include 'charlist.h'
// #include 'cinages.h'
//
// =====
// Scan a line quickly (every 1/4 pixel) looking for an 'on' pixel
// Returns 1 for pixel found, 0 otherwise
// The line to process
// The column to scan
int c;
int r;
for (r=0; r<line.sizeR; r++)
    if (line.pixel[r][c] & 1)
        return 1;
return 0;

// =====
// Scan a line completely looking for an 'on' pixel
// Returns 1 for pixel found, 0 for otherwise
// The line
// The column to scan
int c;
int r;
for (r=0; r<line.sizeR; r++)
    if (line.pixel[r][c] & 1)
        return 1;
return 0;

// =====
// The main routine. Call this to make the list
void extract_isrovp_list(
    cinages line, // The image of the line to segment
    struct Vpnterovplist *list // A pointer to the list
)
{
    int c;
    int left, right;
    for (c=0; c<line.sizeC; c++) // Scan the image
    {
        if (line_scan_quick(line, c) // If found something
        {
            left = right = c; // Look to the left for the start
            while (--c >= 0)
            {
                if (line_scan_complete(line, c) == 0)
                    break;
                left = c;
            }
            c = right; // Keep looking to the right
            while (++c < line.sizeC)
            {
                if (line_scan_complete(line, c) == 0)
                    break;
                right = c;
            }
            isrovp_add(list, left, right); // Add in the new break
        }
    }
}

```

【手続補正書】

【提出日】平成5年5月13日

【手続補正1】

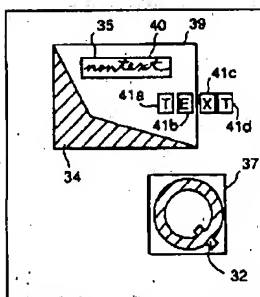
【補正対象書類名】図面

【補正対象項目名】全図

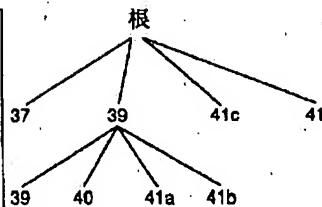
【補正方法】変更

【補正内容】

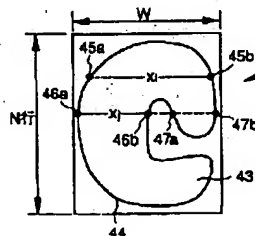
【図5B】



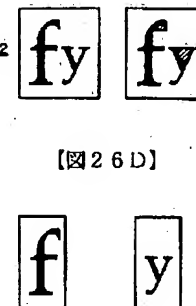
【図5C】



【図6A】

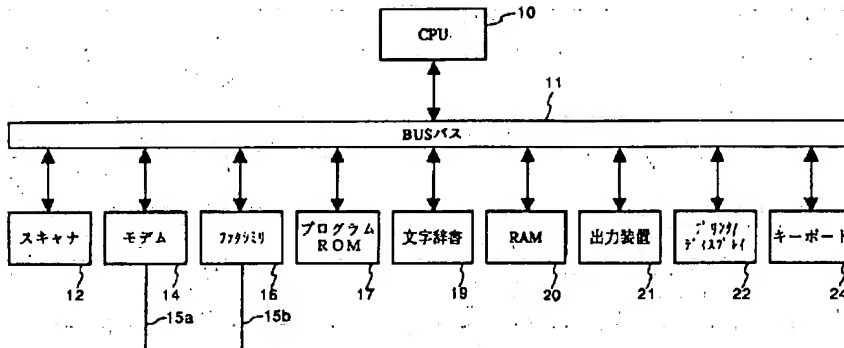


【図26B】【図26C】



【図26D】

【図1】 * *



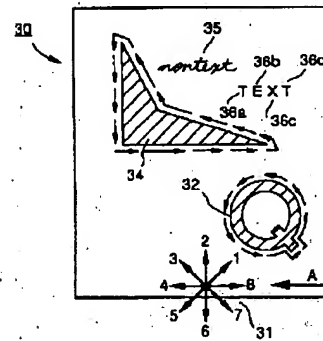
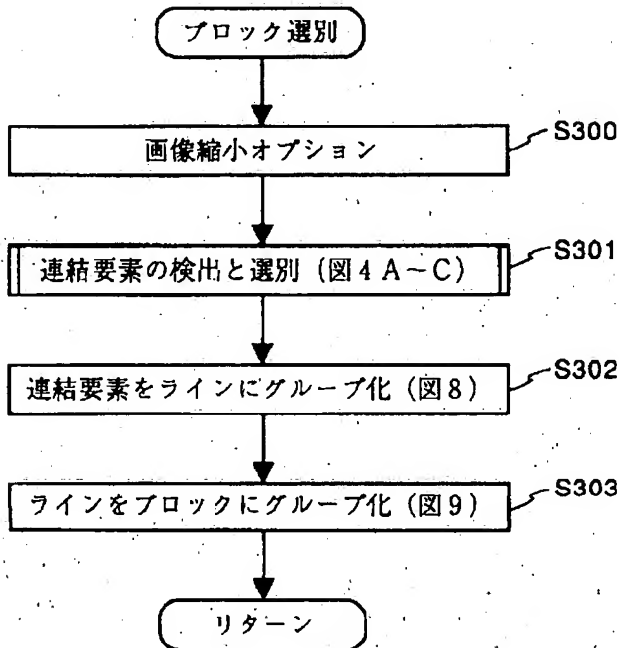
【図35】

Source Code For
Block Selection

【図3】

※ ※

【図5A】



【図18B】

【図234】

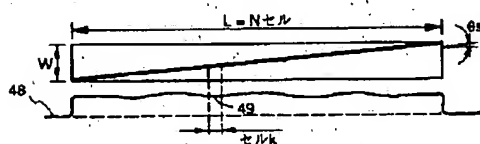


```
free((char **) m);
```

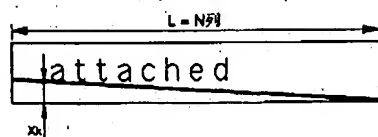
【図6B】

【図6C】

【図194】



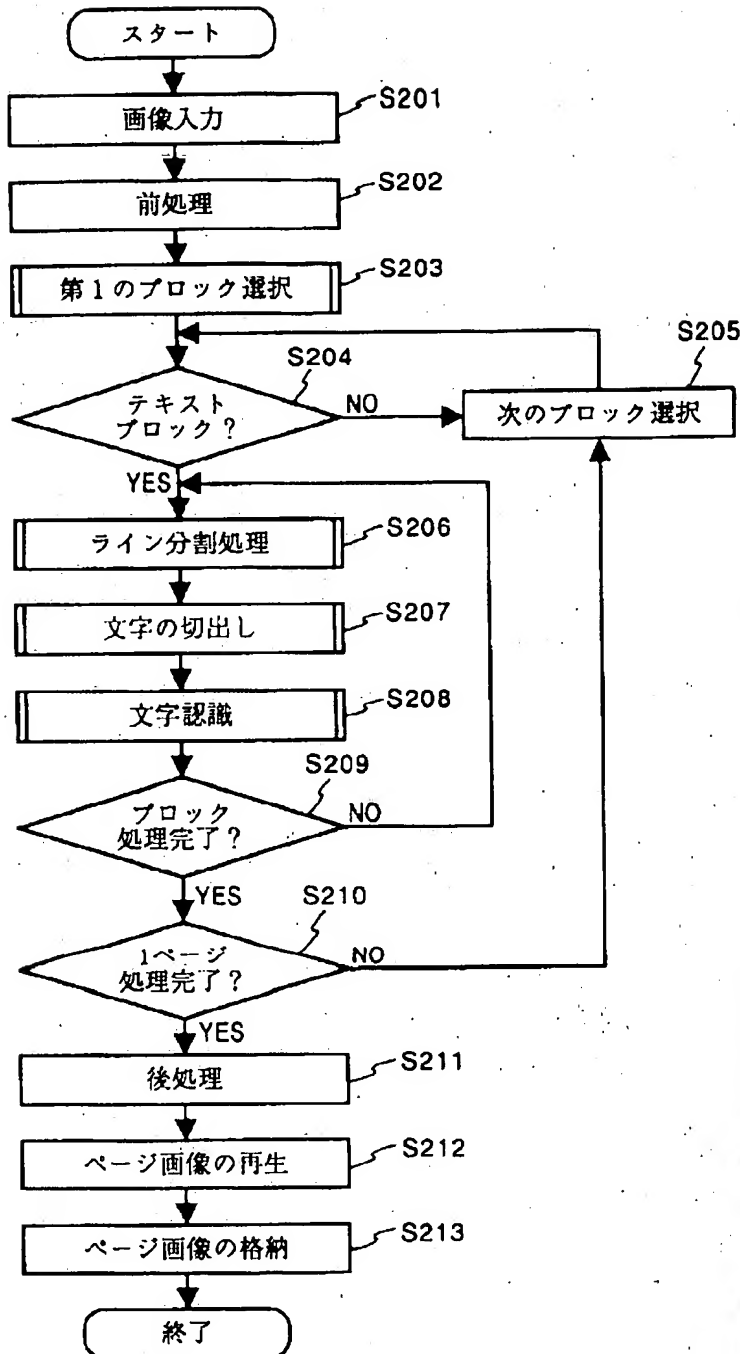
【図126】



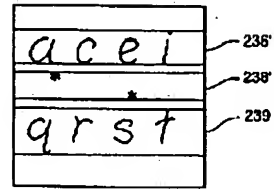
```
free((char **) m);
```

```
#define FILE_LENGTH      80
#define LINE_LENGTH      256
#define PORT_NAME_LENGTH 20
```

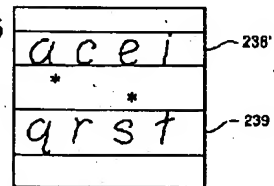

【図2】



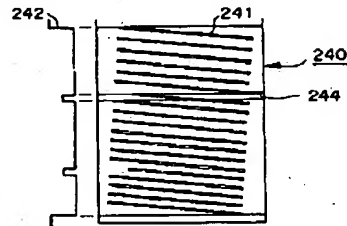
【図18C】



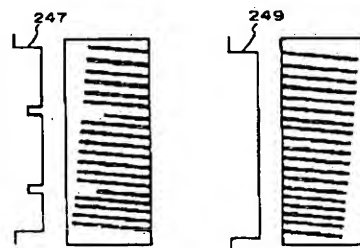
【図18D】



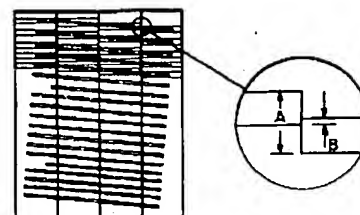
【図19A】



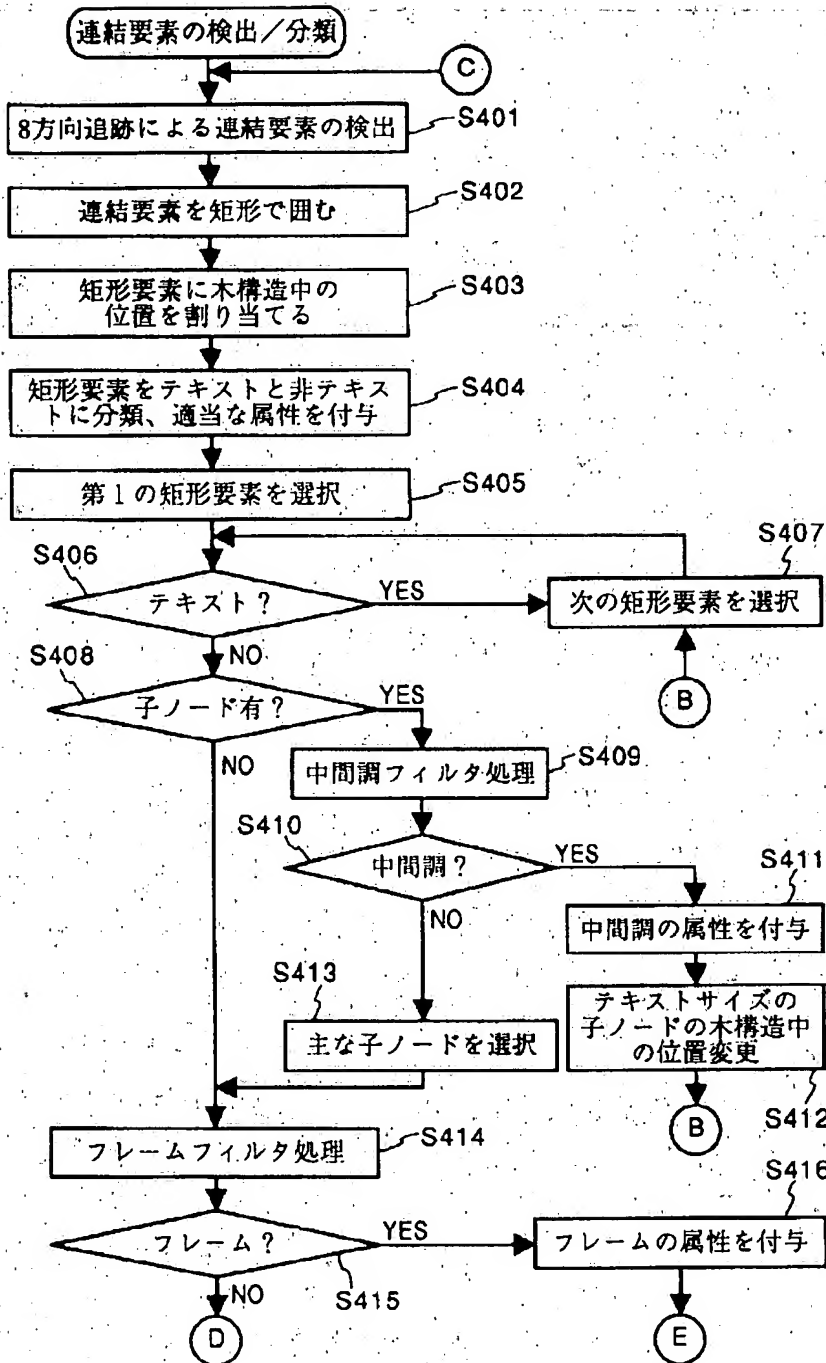
【図19B】



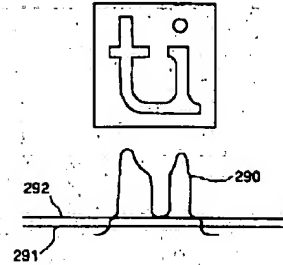
【図19D】



【図4A】



【図29A】



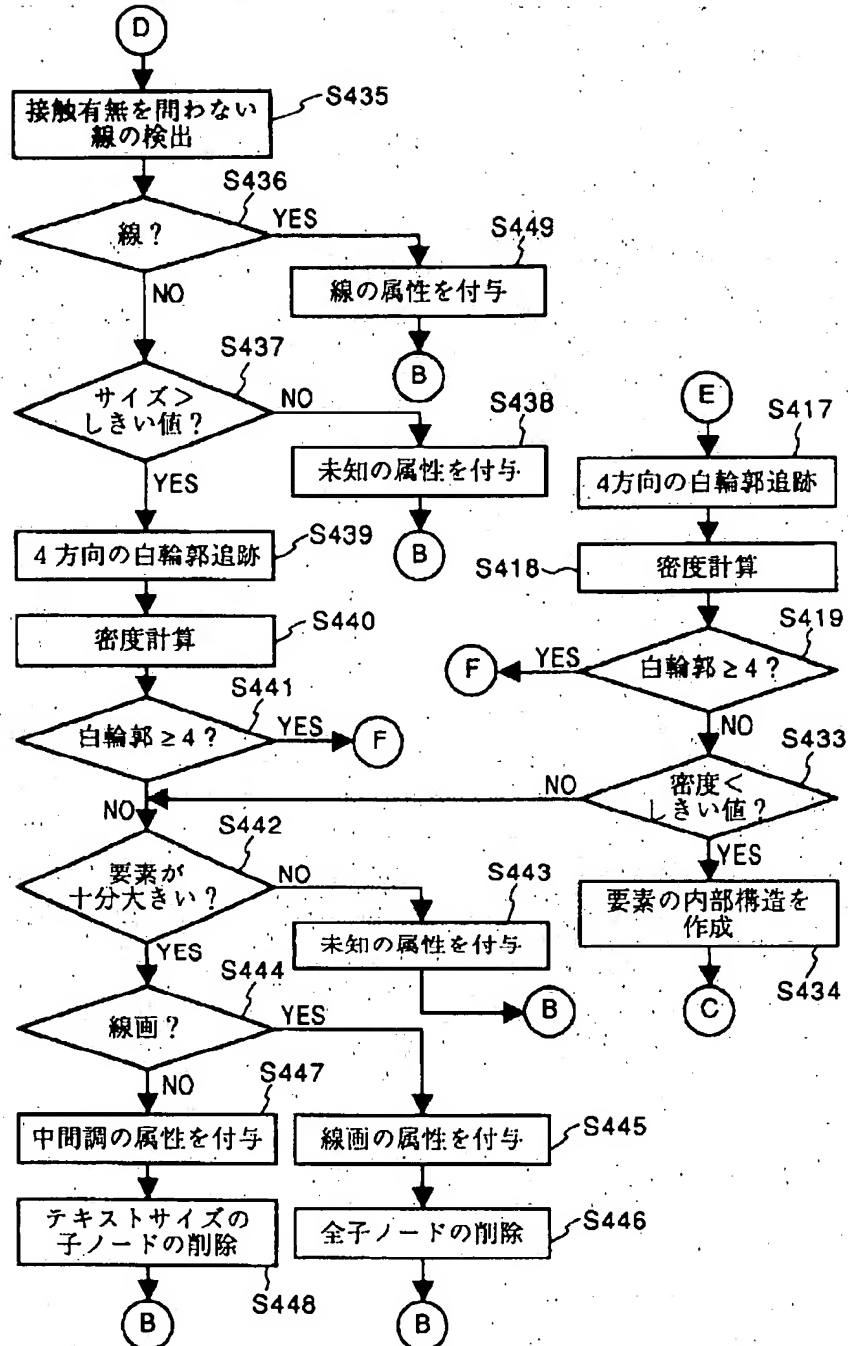
【図134】

```

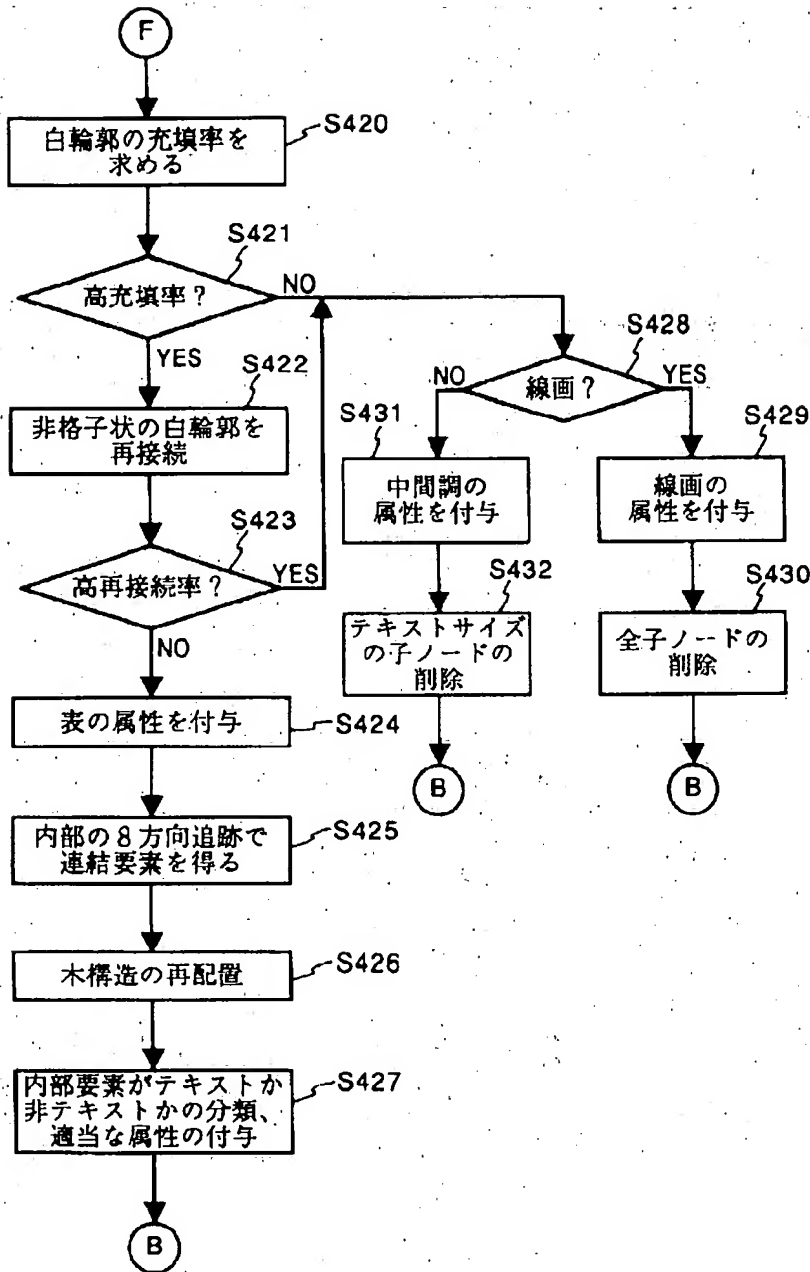
extern float *vector();
extern unsigned char *dvector();
extern int *jvector();
extern double *dvector();
extern unsigned char *cmatrix();
extern float **matrix();
extern int **imatrix();
extern double **dmatrix();
extern void free_vector();
extern void free_dvector();
extern void free_jvector();
extern void free_cmatrix();
extern void free_imatrix();
extern void free_dmatrix();
extern void free_matrix();
extern void free_cmatrix();

```

【図4B】



【図4C】



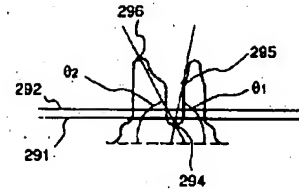
【図26A】

Satisfy

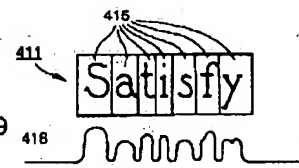
【図34B】

Satisfy

【図29B】



【図34A】



【図163】

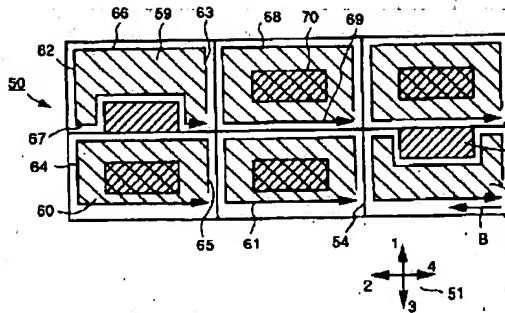
```

HISTOGRAM_THRESHOLD;
charon = charon->next; // Get the next character
free_vector(histogram); // Deallocate the memory

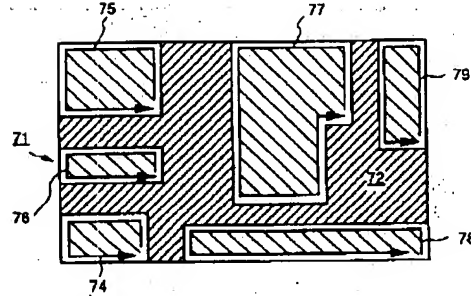
```

【図7A】

* *



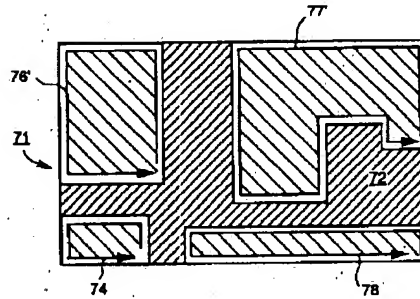
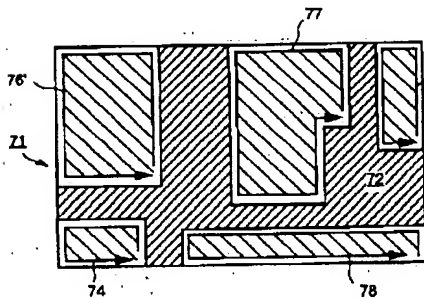
【図7B】



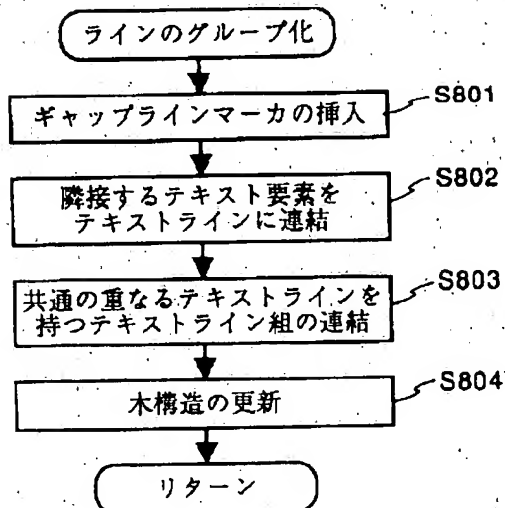
【図7C】

※ ※

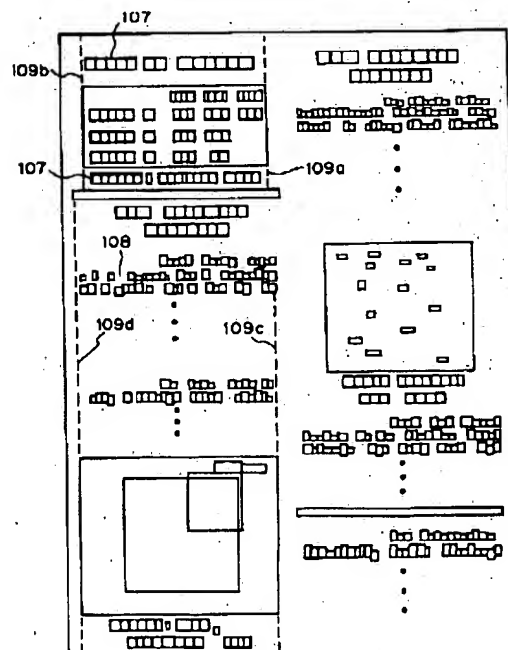
【図7D】



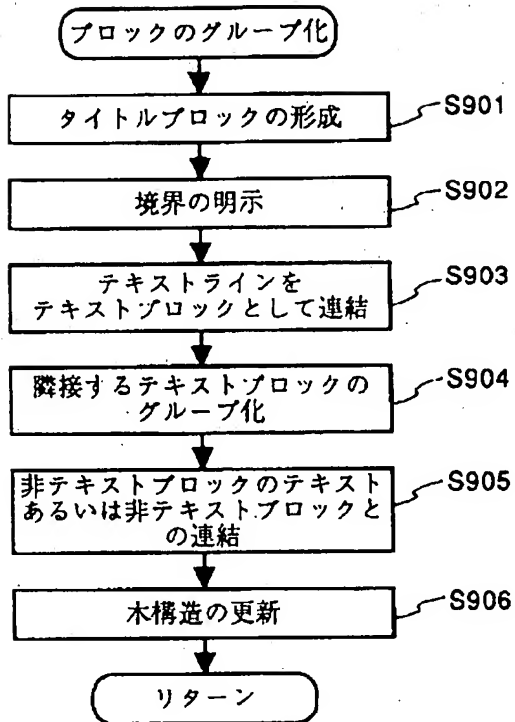
【図8】



【図11】

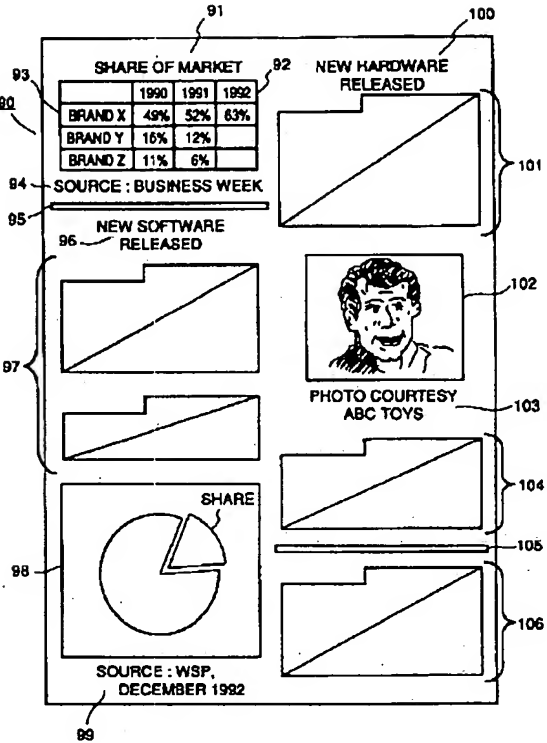


【図9】

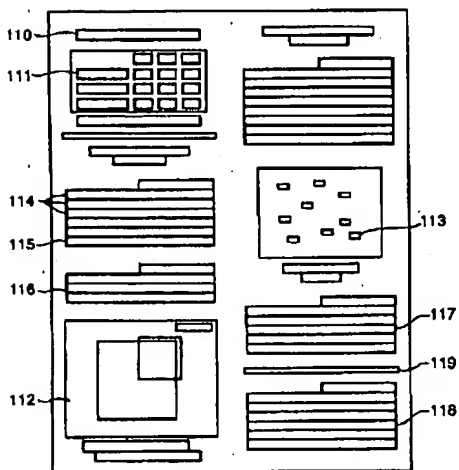


* *

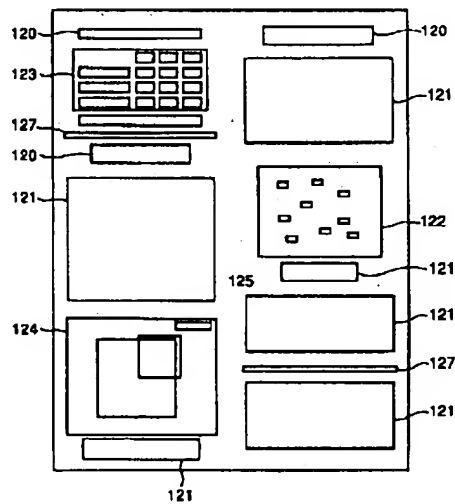
【図10】



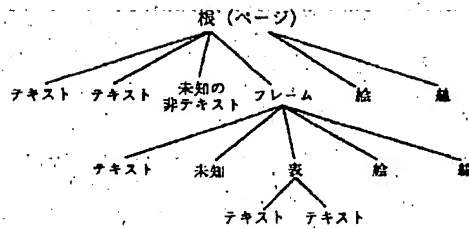
【図12】



【図13】

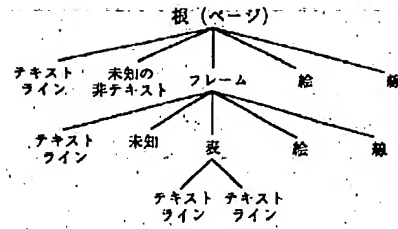


【図14】

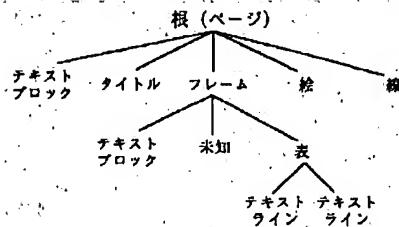


* *

【図15】

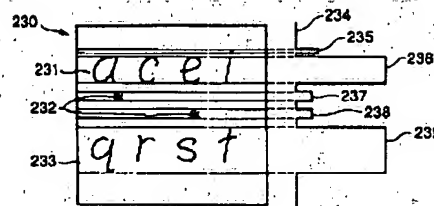


【図16】



※ ※

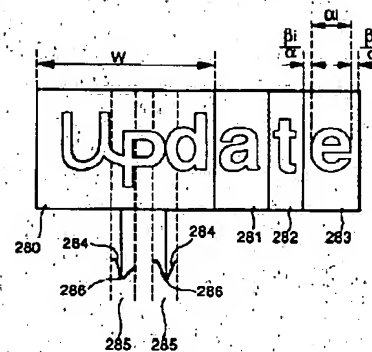
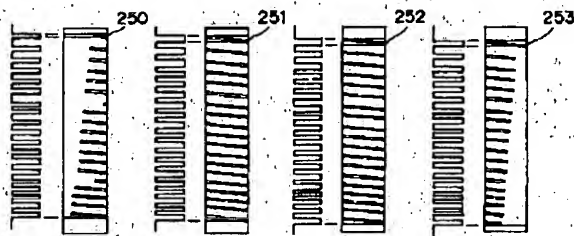
【図18A】



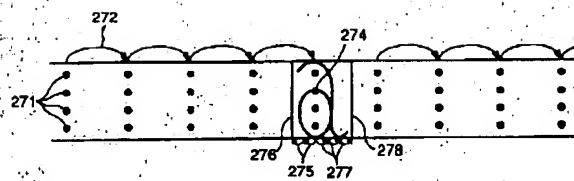
【図19C】

★ ★

【図24】



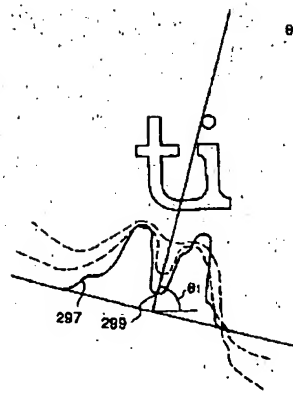
【図22】



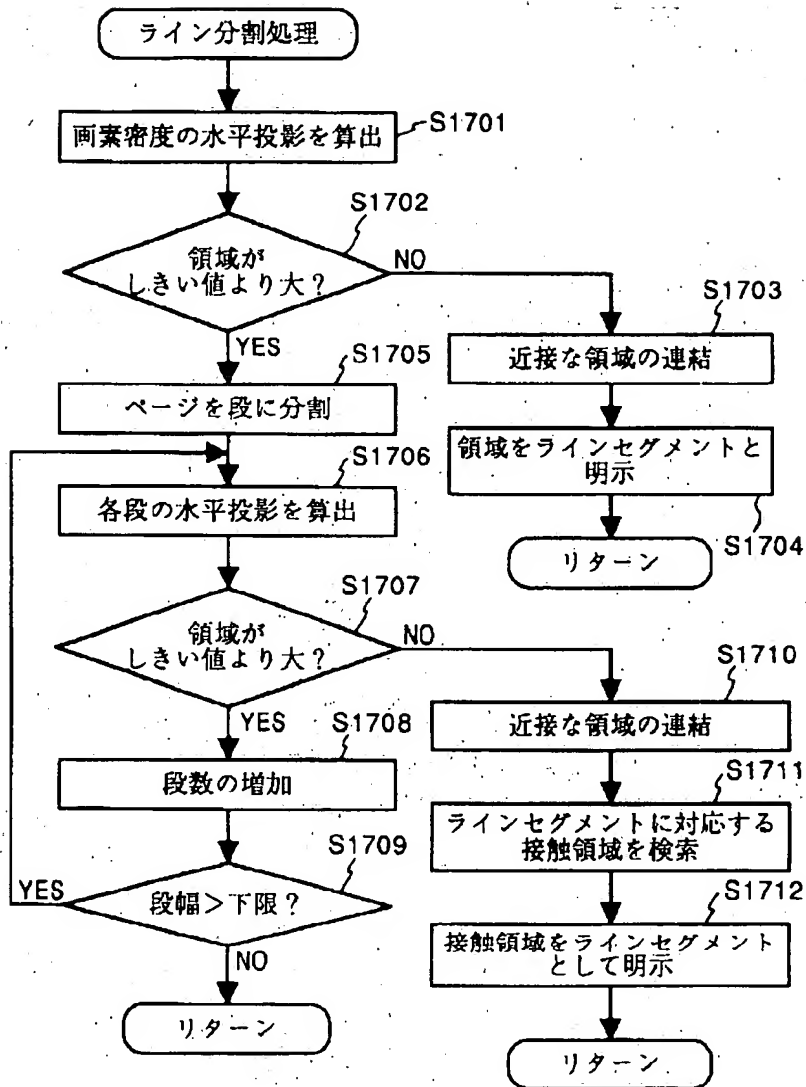
【図29C】

 $81 \pm 3^\circ / \pm 6^\circ$

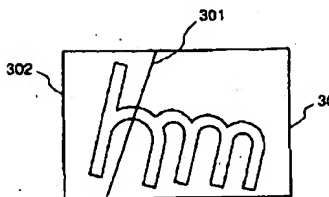
【図29D】

 $81 \pm 3^\circ / \pm 6^\circ$ 

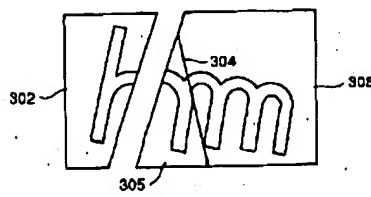
【図17】



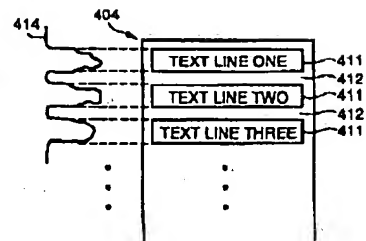
【図31A】



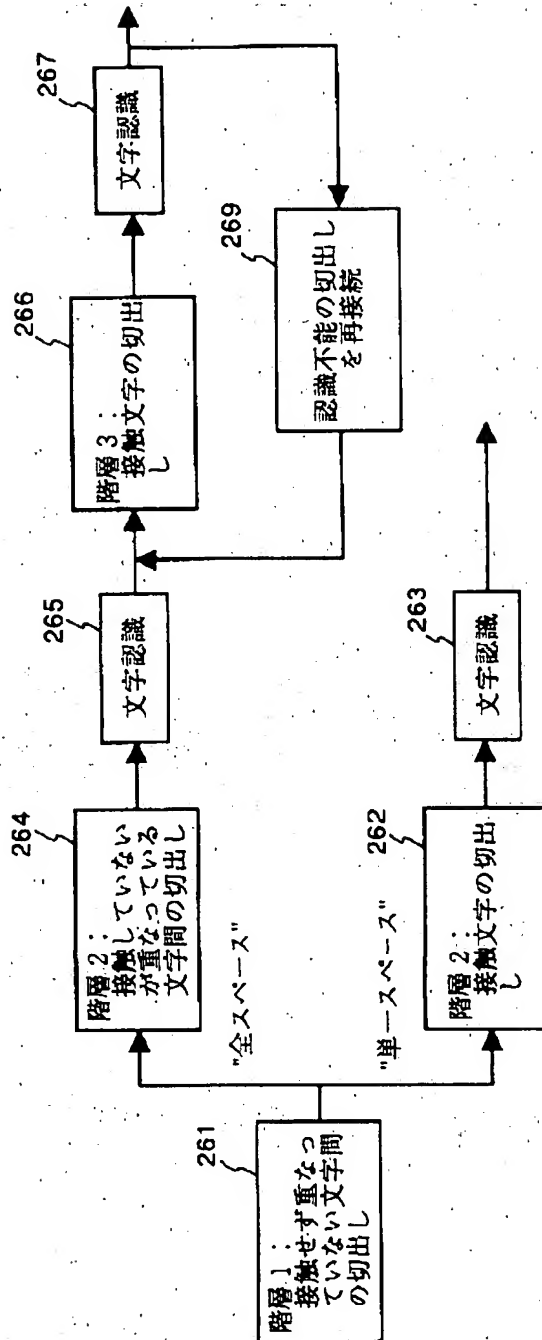
【図31B】



【図33A】



【図20】



【図36】

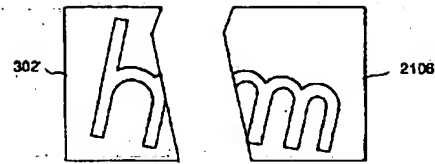
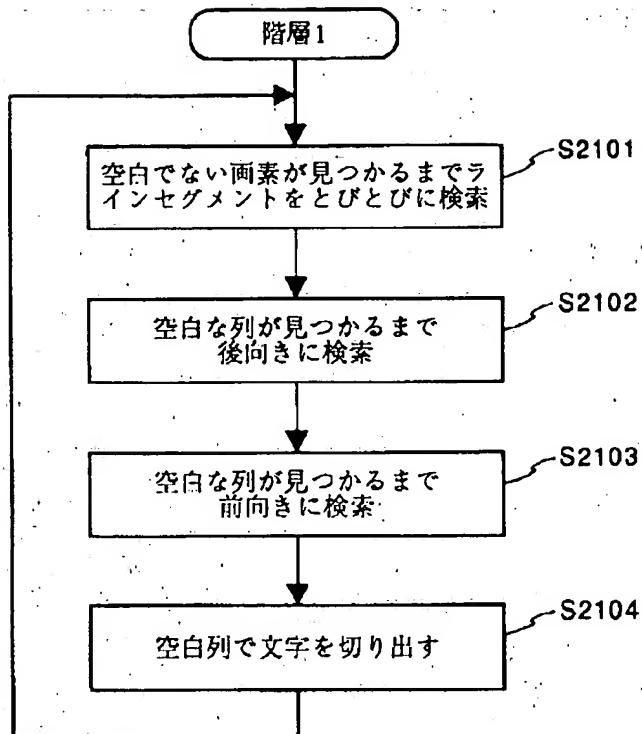
```

#include "util.h"
main(argc,argv)
    unsigned int argc;
    char **argv;
{
    widget w; popup_blockset();
    /*----- Initialise Architect -----*/
    /*----- Initialise Architect -----*/
    Defpanel = Definitilise("OCR Block Selection Demo", argc, argv);
    /*----- Create and popup the first window of the interface. -----*/
    /*----- Create and popup the first window of the interface. -----*/
    w = popup_blockset();
    /*----- Enter the event loop, this function never returns. -----*/
    /*----- Enter the event loop, this function never returns. -----*/
    doMainLoop();
}
  
```

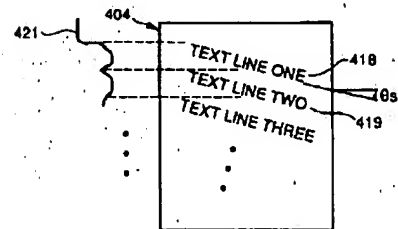
【図21】

* *

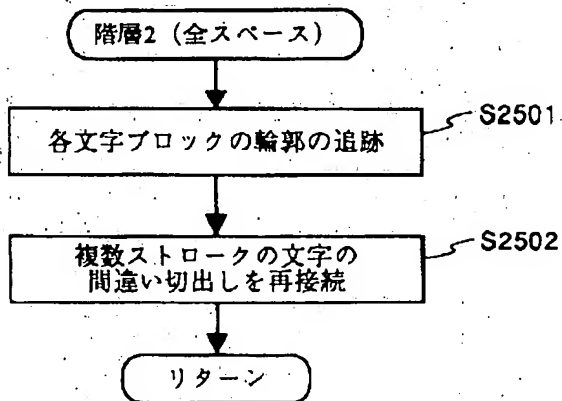
【図31C】



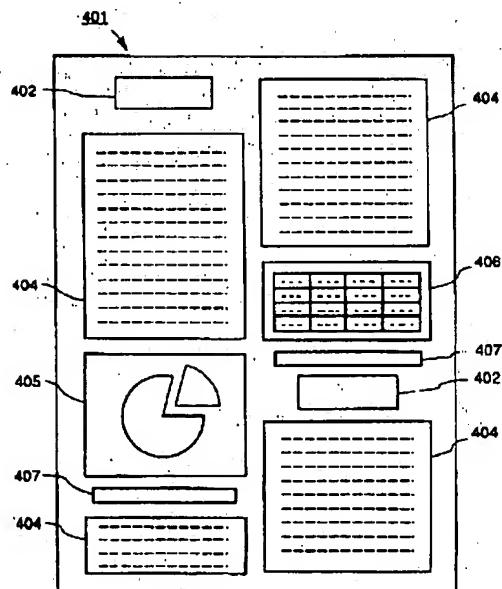
【図33B】



【図25】



【図32】



【図129】

```

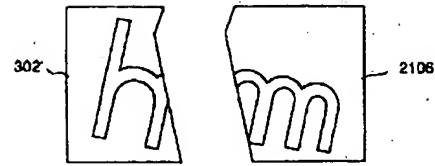
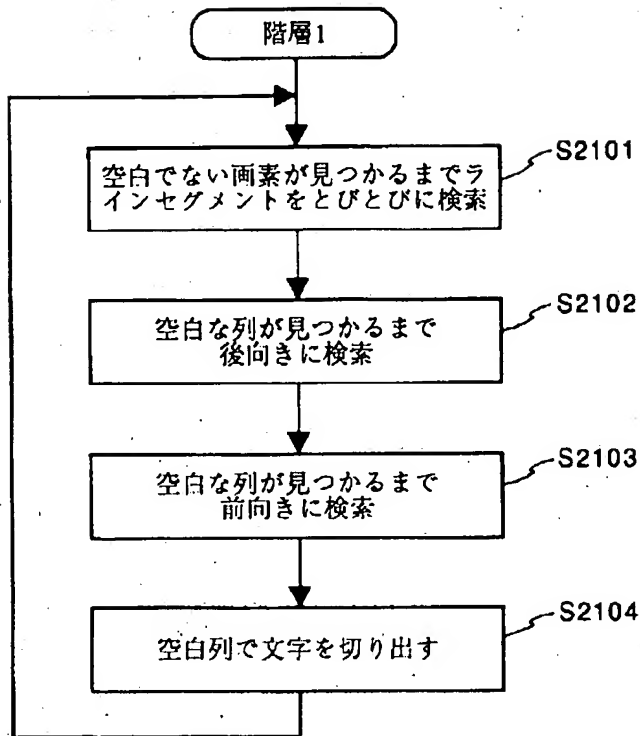
struct sectionblock *previous;
struct sectionblock *next;

int upper_y, lower_y, left_x, right_x;
int width, length;
int index;
int multilineblock;
int att;
int boundary_index;
);
  
```

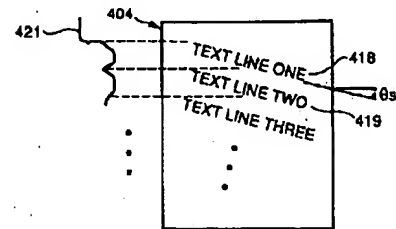
【図21】

*

【図31C】

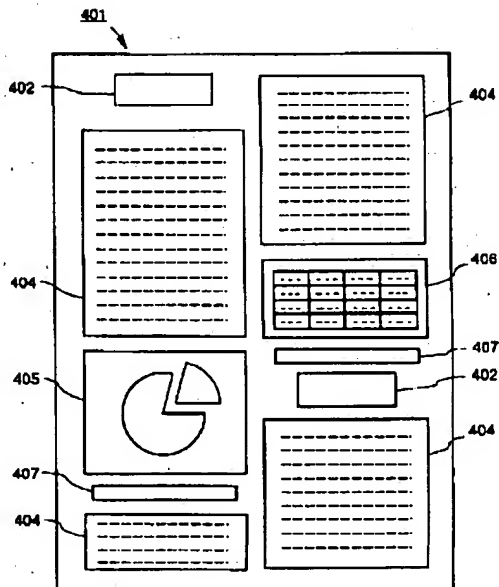
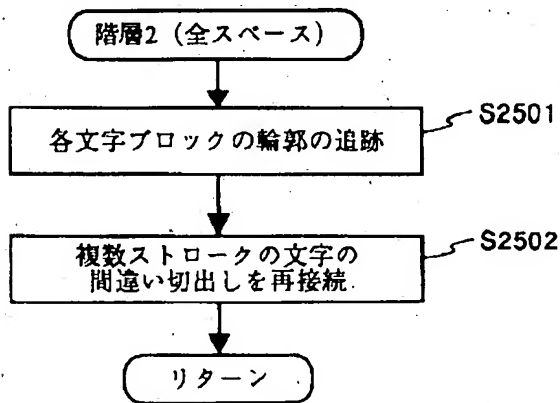


【図33B】



【図25】

【図32】



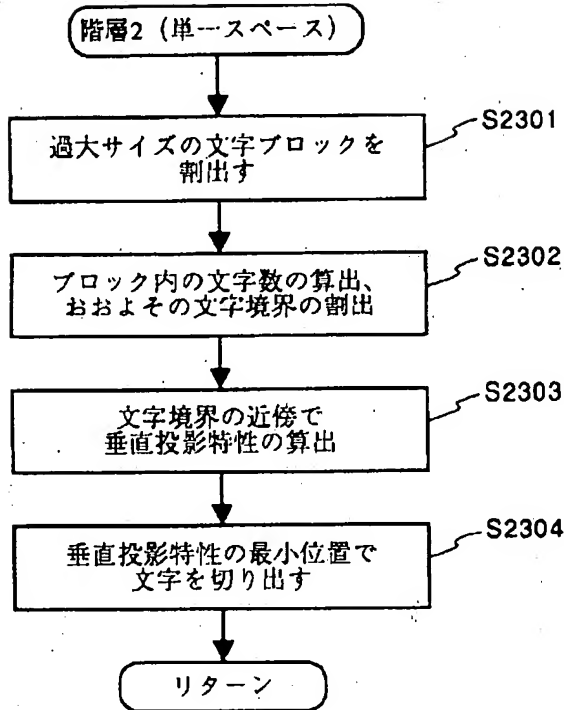
【図129】

```

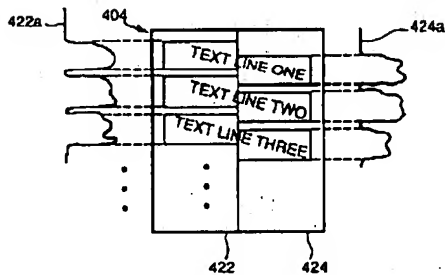
struct sectionblock *previous;
struct sectionblock *next;

int upper_y, lower_y, left_x, right_x;
int width, length;
int index;
int multilineblock;
int att;
int boundary_index;
};
  
```

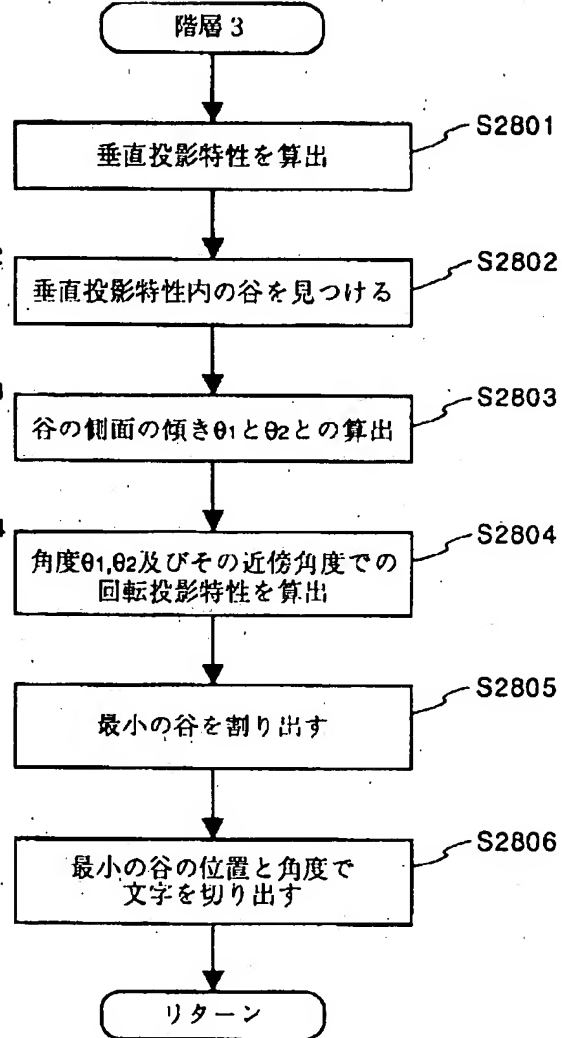
【図23】



【図33C】



【図28】



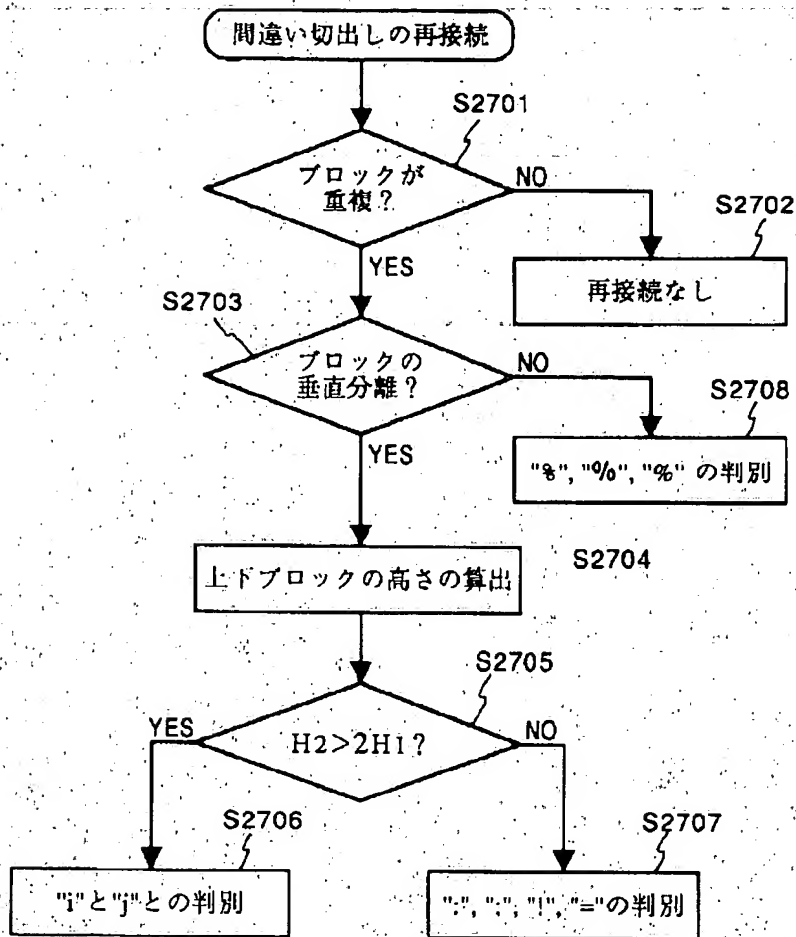
【図44】

```

/* ARGUSED */
static void okCallback_QuitMessageBox(UxWidget, UxContext, UxCallbackArg)
{
    Widget      *pWidget;
    _UxQuitDialog *pQuitDialog;
    int          i;
    UxWidget      UxThisWidget;
    UxWidget      UxWidgetToWidget(UxWidget);
    _UxFromContext(UxContext);
    {
        exit();
    }
    _UxToContext(UxContext);
}

```

【図27】



【図46】

```

_UxCon->DisplayForm= DisplayForm;
_UxCon->DisplayFormClose_FBG= DisplayFormClose_FBG;
_UxCon->DisplayForm_DrawArea= DisplayForm_DrawArea;
}

static void _UxFromContext(_UxCon)
{
    _UxDisplayForm = _UxCon;
    DisplayForm= _UxCon->DisplayForm;
    DisplayFormClose_FBG= _UxCon->DisplayFormClose_FBG;
    DisplayForm_DrawArea= _UxCon->DisplayForm_DrawArea;
}

/* ARGUSED */
static void activateCallback_DisplayFormClose_FBG(UxWidget, UxContext, UxCallbackArg)
{
    UxWidget
    _UxDisplayForm = _UxContext;
    int    UxCallbackArg;
    {
        swidget      UxThisWidget;
        UxThisWidget= UxWidgetToSwidget(UxWidget);
        _UxFromContext(UxContext);
        {
            UxPopdownInterface (DisplayForm);
        }
        _UxToContext (UxContext);
    }
}

```

【図133】

```

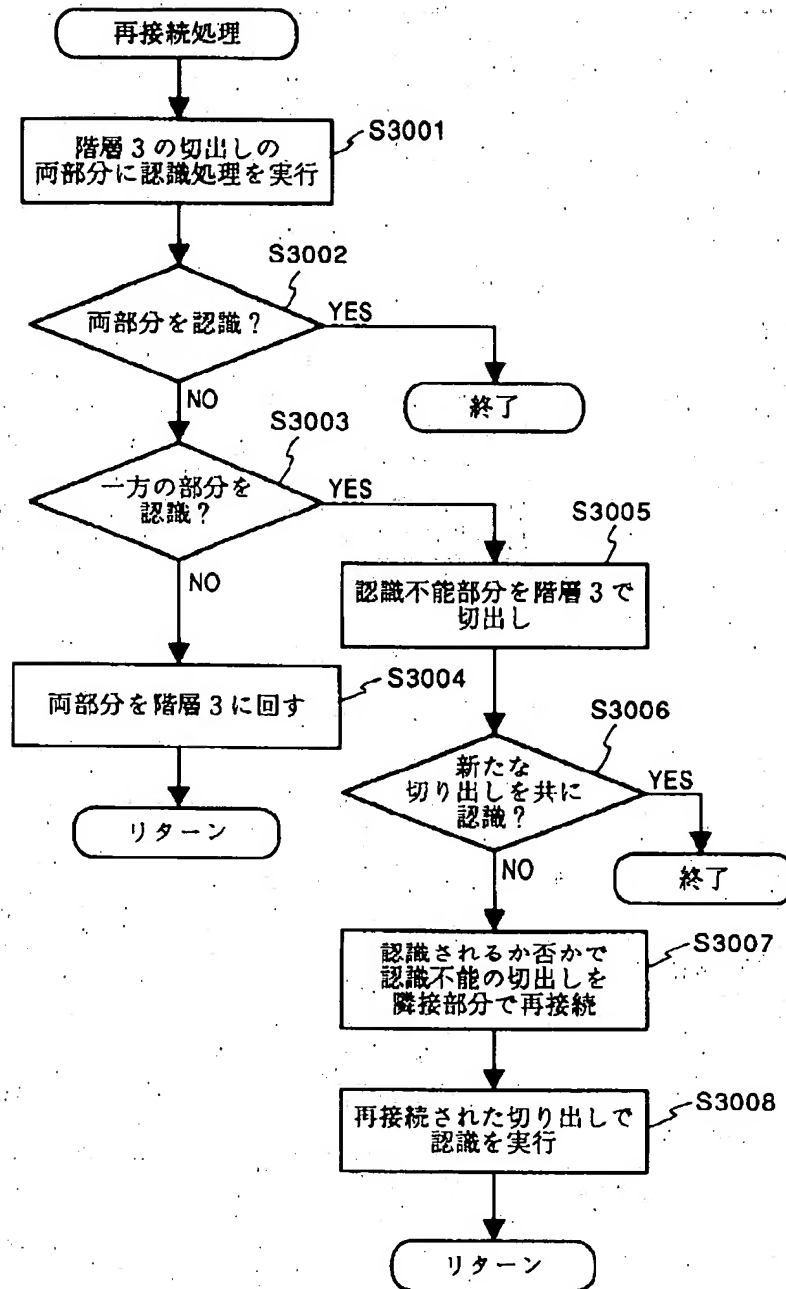
#define R      0
#define G      1
#define B      2

int color_comp[9][3]={
    {0x65, 0xcd, 0x05},
    {0xfe, 0xc0, 0x09},
    {0xfe, 0xc0, 0xfe},
    {0xfe, 0xfe, 0x00},
    {0xfe, 0x98, 0x00},
    {0x98, 0xc0, 0xc0},
    {0x65, 0xfe, 0xc0},
    {0x65, 0xc0, 0xfe},
    {0x4b, 0x4e, 0x5e}};

/*
int color_comp[9][3]={
    {0x00, 0xc0, 0xc0},
    {0x00, 0xc0, 0xc0},
    {0x00, 0xc0, 0xc0},
    {0x00, 0xc0, 0xc0},
    {0x00, 0xc0, 0xc0},
    {0x00, 0xc0, 0xc0},
    {0x00, 0xc0, 0xc0},
    {0x00, 0xc0, 0xc0},
    {0x00, 0xc0, 0xc0}};
*/

```

【図30】



—243—

[illegible]

—244—

[illegible]

【図 39】

[illegible]

【図40】

```

error_jw_subproc ();
return;
}

_UXContext(UXContext);

/* ARGSUSED */
static void activate_callback_activation(UXWidget, UXContext, UXCallbackArg)
Widget widget;
_UXContext(UXContext);
int int;
void void;
_UXWidget(UXWidget);
_UXWidget(UXWidget);
_UXWidget(UXWidget);
{
    UXPopupInterface(QuitDialog, no_grab);
}
_UXContext(UXContext);

/* ARGSUSED */
static void activate_callback_activation(UXWidget, UXContext, UXCallbackArg)
Widget widget;
_UXContext(UXContext);
int int;
void void;
_UXWidget(UXWidget);
_UXWidget(UXWidget);
_UXWidget(UXWidget);
{
    UXPopupInterface(QuitDialog, no_grab);
}
_UXContext(UXContext);

/* Start execution of your program here */
UXPopupInterface(DisplayForm, no_grab);
height = UXWidgetHeight(DisplayForm_DrawArea);
printf ("height == %d\n", height);
width = UXWidgetWidth(DisplayForm_DrawArea);
printf ("width == %d\n", width);
foreground = UXWidgetForeground(DisplayForm_DrawArea);
background = UXWidgetBackground(DisplayForm_DrawArea);
printf ("f = %s b = %s\n", foreground, background);
block_process(UXWidget(DisplayForm_DrawArea), width, height);
_UXContext(UXContext);

/* ARGSUSED */
static void activate_callback_activation(UXWidget, UXContext, UXCallbackArg)
Widget widget;
_UXContext(UXContext);
int int;
void void;
_UXWidget(UXWidget);
_UXWidget(UXWidget);
_UXWidget(UXWidget);
{
    UXPopupInterface(QuitDialog, no_grab);
}
_UXContext(UXContext);

/* color_reference(UXWidget(UXFormArea), width, height);
_UXContext(UXContext);
*/

```

-247-

[illegible]

【図42】

```

_UxCon->FileSelectionDialog= FileSelectionDialog;
_UxCon->FileSelectionBox= FileSelectionBox;
_UxCon->parent= parent;

static void _UxFromContext(_UxCon)
{
    _UxFileSelectionDialog = _UxCon;
    FileSelectionDialog= _UxCon->FileSelectionDialog;
    FileSelectionBox= _UxCon->FileSelectionBox;
    parent= _UxCon->parent;

/* ARGUSED */
static void cancelCallback_FileSelectionBox(UxWidget, UxContext, UxCallbackArg)
{
    Widget      UxWidget;
    _UxFileSelectionDialog = _UxContext;
    int          UxCallbackArg;
    {
        sWidget      UxThisWidget;
        UxThisWidget= UxWidgetToSWidget(UxWidget);
        _UxFromContext(UxContext);
        {
            UxPopdownInterface(FileSelectionDialog);
        }
        _UxToContext(UxContext);
    }

/* ARGUSED */
static void okCallback_FileSelectionBox(UxWidget, UxContext, UxCallbackArg)
{
    Widget      UxWidget;
    _UxFileSelectionDialog = _UxContext;
    int          UxCallbackArg;
    {
        sWidget      UxThisWidget;
        UxThisWidget= UxWidgetToSWidget(UxWidget);
        _UxFromContext(UxContext);

        handle_tiff_handle;
        UxPopdownInterface(FileSelectionDialog);
        strcpy(filename, get_input_filename());
        printf("Open [%s]\n", filename);
        tiff_handle = UxCreatSubproc("/usr/local/bin/X11/imageview", filename,
UxAppendTo);
        if (tiff_handle == -1) {
            error_create_subproc();
            return;
        }
        if (UxRunSubproc(tiff_handle, NULL) == -1) {
            error_run_subproc();
            return;
        }
        open_tiff_file(filename);
        _UxToContext(UxContext);
    }
}

```

【図48】

```

static void _UxFromContext(_UxCon)
{
    _UxCInfoForm = _UxCon;
    InfoForm= _UxCon->InfoForm;
    InfoFormClose_PBG= _UxCon->InfoFormClose_PBG;
    InfoFormArea= _UxCon->InfoFormArea;

/* ARGUSED */
static void activateCallback_InfoFormClose_PBG(UxWidget, UxContext, UxCallbackArg)
{
    Widget      UxWidget;
    _UxCInfoForm = _UxContext;
    int          UxCallbackArg;
    {
        sWidget      UxThisWidget;
        UxThisWidget= UxWidgetToSWidget(UxWidget);
        _UxFromContext(UxContext);
        {
            UxPopdownInterface(InfoForm);
        }
        _UxToContext(UxContext);
    }
}

```

【図62】

```

int Width, height;
{
    int x, y, xdis, ydis, i;
    static fid = 33;
    init_graphics(wd);
    xs_clr_vin(wd);
    xdis = ((height-20)/(TOTALCOLOR));
    xs_getfont(wd);
    for ( i = 0; i < TOTALCOLOR; i++){
        xs_set_foreground(color_code[i], wd);
        xs_setfont(wd, fid);
        xs_string(wd, 10, (i+1)*xdis+10, color_label[i]);
    }
    xs_flush(wd);
}

```

—249—

[illegible]

—250—

[illegible]

—251—

[illegible]

—252—

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <sys/time.h>

struct timeval first, second, lapued;
struct timespec ttp;

/* Start the timing clock */
start_clock ()
{
    gettimeofday (&first, &ttp);
}

/* Stop the timing clock and return the elapsed time in milliseconds. */
double stop_clock ()
{
    double time;

    gettimeofday (&second, &ttp);
    if (first.tv_usec > second.tv_usec) {
        second.tv_usec -= 1000000;
        second.tv_usec += first.tv_usec;
    }
    time = (double)((second.tv_sec - first.tv_sec) * 1000 +
        (double)((second.tv_usec - first.tv_usec) / 1000));
    printf ("*** Elapsed time as %.15th sec.\n", time);
    return time;
}

/* Return the basename of a file without the directory */
char *extract_base_file_name (name)
{
    char *base;

    char cap(FILE_LENGTH), *pos;
    strcpy (cap, name);
    if ((pos = strchr(cap, '/')) != NULL)
        return pos+1;
    else
        return name;
}

/* Return the basename of a file */
char *extract_base_file (name)
{
    char *base;

    char cap(FILE_LENGTH), *pos;
    strcpy (cap, name);
    if ((pos = strchr(cap, '/')) != NULL)
        return pos+1;
    else
        return name;
}

/* Return -1 if file cannot open */
int check_file (name)
{
    char *base;
    FILE *fp;

    if ((fp = fopen (name, "r")) == NULL)
        return -1;
    fclose (fp);
    return 1;
}

/* Print error message */
print_error (messg)
{
    char *messg;

    printf ("***Error: %s\n", messg);
}

/* Create a subprocess */
error_create_subproc ()
{
    printf ("***Error: Cannot create a subprocess!\n");
}

/* Subprocess exit callback */
error_subproc_exit_callback ()
{
    printf ("***Error: Cannot set subprocess exit callback!\n");
}

/* Subprocess */
error_run_subproc ()
{
    printf ("***Error: Cannot start a subprocess!\n");
}

```


—253—

[illegible]

【図51】

```

        XSetForeground(XtDisplay(wd), drawing_gc, color);
        XtSetString(XtDisplay(wd), XtWindow(wd), drawing_gc, x, y, str, strlen(str));
        XSetForeground(XtDisplay(wd), drawing_gc, foreground);
    }

    static char *font;
    int font_wd;
    int font_hd;

    {
        int i, num = 53;
        font = XListFonts(XtDisplay(wd), "", 50, &num);
        printf("num = %d\n", num);
    }

    XSetFont(wd, fid)
    widget wd;
    int fid;
    {
        XFontStruct *font_info;
        if ((font_info = XQueryFont(XtDisplay(wd), font[fid]) == NULL) {
            printf("can not open the font fid = %d\n", fid);
            exit(-1);
        }
        XSetFont(XtDisplay(wd), drawing_gc, font_info->fid);
        printf("fid = %d\n", fid);
    }

    *height = win_height;
}

XtDrawLine(wd, x1, y1, x2, y2)
Widget wd;
int x1, y1, x2, y2;
{
    XDrawLine(XtDisplay(wd), XtWindow(wd), drawing_gc, x1, y1, x2, y2);
}

xs_color_point(wd, x, y, color)
Widget wd;
int x, y, color;
{
    Arg args[1];
    int foreground;
    XSetArg(args[0], XtForegroundColor, foreground);
    XtGetValues(wd, args, 1);
    XSetForeground(XtDisplay(wd), drawing_gc, color);
    XDrawPoint(XtDisplay(wd), XtWindow(wd), drawing_gc, x, y);
    XSetForeground(XtDisplay(wd), drawing_gc, foreground);
}

xs_color_line(wd, x1, y1, x2, y2, color)
Widget wd;
int x1, y1, x2, y2, color;
{
    Arg args[1];
    int foreground;
    XSetArg(args[0], XtForegroundColor, foreground);
    XtGetValues(wd, args, 1);
    XSetForeground(XtDisplay(wd), drawing_gc, color);
    XDrawLine(XtDisplay(wd), XtWindow(wd), drawing_gc, x1, y1, x2, y2);
    XSetForeground(XtDisplay(wd), drawing_gc, foreground);
}

xs_color_box(wd, x1, y1, x2, y2, color)
Widget wd;
int x1, y1, x2, y2, color;
{
    Arg args[1];
    int foreground;
    XSetArg(args[0], XtForegroundColor, foreground);
    XtGetValues(wd, args, 1);
    XSetForeground(XtDisplay(wd), drawing_gc, color);
    XDrawRectangle(XtDisplay(wd), XtWindow(wd), drawing_gc, x1, y1, x2, y2);
    XSetForeground(XtDisplay(wd), drawing_gc, foreground);
}

xs_color_str(wd, x, y, str, color)
Widget wd;
int x, y, color;
char *str;
{
    Arg args[1];
    int foreground;
    XSetArg(args[0], XtForegroundColor, foreground);
    XtGetValues(wd, args, 1);
    XSetForeground(XtDisplay(wd), drawing_gc, color);
    XDrawTextString(XtDisplay(wd), XtWindow(wd), drawing_gc, x, y, str);
    XSetForeground(XtDisplay(wd), drawing_gc, foreground);
}

```

—255—

[illegible]

【図53】

```

printf("\nError in opening %s\n", filename);
exit(1);
tagfile_position = 0;
/* Read in header and make sure it's Intel */
i = getbytes(tagfile, tagfile_position);
if (i != 0x4949)
{
    printf("\nError: TIFF file not in Intel Format\n");
    exit(1);
}
/* Read in Magic number
i = getbytes(tagfile, tagfile_position);
if (i != 0x002a)
{
    printf("\nBAD Magic Number for TIFF file!\n");
    exit(1);
}
/* Find First File Directory
j = getbytes(tagfile, tagfile_position);
while(tagfile_position < j)
{
    i = getbytes(tagfile, tagfile_position);
}
/* This is Directory
i = getbytes(tagfile, tagfile_position);
while (i != 0)
{
    tag = getbytes(tagfile, tagfile_position);
    type = getbytes(tagfile, tagfile_position);
    len = getbytes(tagfile, tagfile_position);
    value = getbytes(tagfile, tagfile_position);
    if ((len != 1) && ((tag & 32768) == 0))
    {
        printf("Error: Tag Length != 1\n");
        exit(1);
    }
    printf("tag = %x\n", tag);
    switch (tag)
    {
        case 0x0001: /* NewSubFileType */
            if (value != 0)
            {
                printf("Can't Deal with Non-Standard NewSubFileType\n");
                exit(1);
            }
            break;
        case 0x0002: /* NewSubFileType */
            if (value != 1)
            {
                printf("Subfile Type not full resolution\n");
                exit(1);
            }
            break;
        case 0x0003: /* Width */
            width = value;
            break;
        case 0x0004: /* Length */
            length = value;
            break;
    }
}

```

```

case 0x0005: /* Bits per sample */
    if (value != 1)
    {
        printf("Error: Bits/sample must be 1\n");
        exit(1);
    }
    break;
case 0x0006: /* Compression */
    if (value != 1)
    {
        printf("Compression Used! Error\n");
        exit(1);
    }
    break;
case 0x0007: /* Black is */
    if (value != 1)
    {
        printf("Thresholding tag\n");
        exit(1);
    }
    break;
case 0x0008: /* Offset to Aster Data */
    raster_offset = value;
    break;
case 0x0009: /* Rows = value; */
    rows = value;
    break;
case 0x000a: /* Columns = value; */
    columns = value;
    break;
case 0x000b: /* Rows per strip = value; */
    rows_per_strip = value;
    break;
case 0x000c: /* Strips per image = value; */
    strips_per_image = value;
    break;
default:
    if ((value & 32768) == 0)
    {
        printf("Warning: An unsupported tag appears in the TIFF file\n");
        break;
    }
    while(tagfile_position < raster_offset)
    {
        i = getbytes(tagfile, tagfile_position);
    }
    /*----- END IN FILE -----*/
    all_length = (int)((columns*rows)/SCALE_RATIO);
    all_width = (int)((width*rows)/SCALE_RATIO);
    /* printf("all_length = %d\n", all_length); */
    /* printf("all_width = %d\n", all_width); */
    page = image_allocate((int) all_length, (int) all_width);
    printf("page-size = %d\n", page-size);
    printf("black = %d\n", black);
    for (i = 0; i < all_length; i++)
    {
        for (j = 0; j < all_width; j++)
        {
            if (i < all_length)
            {
                v = getc(tagfile);
                tagfile_position++;
            }
        }
    }
}

```

【図54】

```

if (v == EOP)
{
    printf("\n Error: Premature End of File found in tiff file\n");
    exit(1);
}
if ((v & 128) == black*128)
    page->pixel[i][j] = 0;
else
    page->pixel[i][j] = 1;
v = v << 1;
}
fclose(tagfile);
return page;
}

```

【図86】

```

        release_block(tmp1);
        tmp1 = tmp11;
        for (; tmp1 != NULL; ) {
            tmp12 = tmp1->next;
            release_block(tmp1);
            tmp1 = tmp12;
        }
        free_rectblock(initial_block);
        return;
}
/* Draw text components for display */
void block_circumulate(p, width, height, p_ori)
unsigned char **p;
int width, height;
unsigned char **p_ori;
{
    int i, j;
    /* open file for storing block data */
    original_p = p_ori;
    reduce_factor = SCALE_RATIO;
    for (i = 0; i <= 8; i++)
        for (j = 0; j <= 8; j++)
            direct_stat[i][j] = 0;
    block_file = fopen("block_file.dat", "w");

    text_length = 150.0/SCALE_RATIO;
    total_text = 0;

    /* the root block stands for the whole page image unit */
    root_rectblock = new_block(0,0);
    root_rectblock->width = width;
    root_rectblock->length = height;
    root_rectblock->upper_y = 0;
    root_rectblock->lower_y = height-1;
    root_rectblock->left_x = 0;
    root_rectblock->right_x = width-1;

    /* find blocks inside the page range */
    block_in_range(p, 0, 0, width-1, height-1, root_rectblock);
    printf("get out of block_in_range\n");
    rearrange_rectblock(root_rectblock);
    printf("get out of rearrange_rectblock\n");
    text_discriminate(root_rectblock);
    printf("get out of text_dis\n");
    firstlevelnontextsearch(p, root_rectblock);
    printf("get out of firstlevelnontextsearch\n");
    print_block(1, root_rectblock);

    fclose(block_file);
    printf("before get out of block_construct\n");
}

```

【図137】

Source Code for Proportional Spaced Segmentation

—258—

[illegible]

—259—

[illegible]

【図57】

```

for (tapi = firstline->firstsubline, tapi != NULL, tapi = tapi->next) {
    x1 = (int)((float)tapi->upper_x/SCALE_DISPLAYFORM);
    x2 = (int)((float)tapi->upper_y/SCALE_DISPLAYFORM);
    y1 = (int)((float)tapi->width/SCALE_DISPLAYFORM);
    y2 = (int)((float)tapi->length/SCALE_DISPLAYFORM);
    if (y2 < 1) y2 = 1;
    if (y1 < 1) y1 = 1;
    if (tapi->picture) {
        x1_draw_line(x1, y1, x2, y2);
        break;
    }
}

for (tapi = firstline->firstsubline, tapi != NULL, tapi = tapi->next) {
    x1 = (int)((float)tapi->upper_x/SCALE_DISPLAYFORM);
    x2 = (int)((float)tapi->upper_y/SCALE_DISPLAYFORM);
    y1 = (int)((float)tapi->width/SCALE_DISPLAYFORM);
    y2 = (int)((float)tapi->length/SCALE_DISPLAYFORM);
    if (y2 < 1) y2 = 1;
    if (y1 < 1) y1 = 1;
    if (tapi->picture) {
        x1_draw_line(x1, y1, x2, y2);
        break;
    }
}

//***** Display the lineblock
//*****

void display_sectionblock(firstsection, wd)
struct sectionblock *firstsection;
Widget wd;

{
    struct sectionblock *tapi;
    int x1, y1, x2, y2;

    if (firstsection->atttable)
        x1_set_foreground(FGTEXT_COLOR, wd);
    else
        x1_set_foreground(FGTEXT_COLOR, wd);
    for (tapi = firstsection->firstsubsection, tapi != NULL, tapi = tapi->next) {
        x1 = (int)((float)tapi->upper_x/SCALE_DISPLAYFORM);
        y1 = (int)((float)tapi->upper_y/SCALE_DISPLAYFORM);
        y2 = (int)((float)tapi->width/SCALE_DISPLAYFORM);
        if (y2 < 1) y2 = 1;
        if (y1 < 1) y1 = 1;
        x1_draw_box(x1, y1, x2, y2);
        if (tapi->atttable) {
            if (tapi->picture) {
                x1_draw_line(x1, y1, x2, y2);
                break;
            }
        }
    }
    if (tapi->atttable)
        x1_set_foreground(FGTEXT_COLOR, wd);
    else
        x1_set_foreground(FGTEXT_COLOR, wd);
}

//***** draw one side of block
//*****

void draw_side(section, wd, color, side)
struct sectionblock *section;
Widget wd;
int color, side;

{
    int x1, x2, y1, y2;

    x1 = (int)((float)section->upper_x/SCALE_DISPLAYFORM);
    x2 = (int)((float)section->upper_y/SCALE_DISPLAYFORM);
    y1 = (int)((float)section->width/SCALE_DISPLAYFORM);
    y2 = (int)((float)section->length/SCALE_DISPLAYFORM);
    x1_set_foreground(color, wd);
    case 1: x1_draw_line(x1, y1, x2, y2);
    break;
    case 2: x1_draw_line(x1, y1, x2, y2);
    break;
}

```


【图 5 8】

[illegible]

—262—

```

0, reduce_image_width-1);

printf("before\n");
if (step == 0) {
    free_matrix_image, 0, reduce_image_height-1,
    0, reduce_image_width-1);
    free_section(firstsectionblock);
    printf("after free section\n");
}

image_malloc(page);
step = 0;

printf("reading....");
page = read_if(filename);
printf("Done\n");
get_fido_header(page->pixel, 1,1, page->size-1, page->size-1);
printf("size = %d size = %d\n", page->size, page->size);
create_reduce_image();
step = SP_JENISP_REDUCE;

//..... get text block into text display
//.....
//.....
//.....
void get_text(textspec, p, wd, color)
struct sectionblock *textspec;
unsigned char *p;
int wd;
{
    struct lineblock *tmp;
    struct block *root *tmp2;
    int x, y, xl, yl;

    x = get_color(color, wd);
    for (tmp = textspec->firstlineblock; tmp != NULL; tmp = tmp->next) {
        for (y = tmp->upper_y; y < tmp->lower_y; y++)
            for (x = tmp->left_x; x < tmp->right_x; x++)
                p[y][x] = TEXT_BLOCK;
        yl = (int)((float)y/SCALE_DISPLAY);
        xl = (int)((float)x/SCALE_DISPLAY);
        x = block_x(wd, xl, yl);
    }
}

void get_tablet(textspec, p, wd)
struct sectionblock *textspec;
unsigned char *p;
{
    int x, y, xl, yl;
}

//..... get text into table text display
//.....
//.....
//.....
void get_tablet(textspec, p, wd)
struct sectionblock *textspec;
unsigned char *p;
{
    int x, y, xl, yl;
}

```

—263—

[illegible]

—264—

[illegible]

—265—

```

//.....
//..... File: malloc.c
//..... Author: Chin-Yuan Wang
//..... Date: 10-17
//..... Version: 1.0
//..... Copyright: 2008 by Chin-Yuan Wang
//..... All rights reserved.
//.....
#include malloc.h
#include stdio.h
void main() {
    char error_text[100];
    while (1) {
        printf("Enter 'q' to quit, 'e' to error, 'v' to vector, 'f' to float, 'd' to double, 'm' to matrix, 'n' to new, 'r' to realloc, 's' to sizeof, 't' to test, 'u' to unset, 'w' to write, 'x' to exit.\n");
        char c;
        if ((c = getchar()) == '\n') continue;
        if (c != 'q' && c != 'e' && c != 'v' && c != 'f' && c != 'd' && c != 'm' && c != 'n' && c != 'r' && c != 's' && c != 't' && c != 'u' && c != 'w' && c != 'x') {
            printf("Invalid input!\n");
            continue;
        }
        switch (c) {
            case 'q': return 0;
            case 'e': {
                unsigned char *error_text;
                int i;
                for (i = 0; i < n; i++) {
                    error_text[i] = '\0';
                }
                printf("Error: %s\n", error_text);
                break;
            }
            case 'v': {
                float *vector;
                int n;
                printf("Allocate a float vector with range [%d..%d]\n", 0, n-1);
                vector = (float *) malloc(sizeof(float) * n);
                if (!vector) {
                    printf("Allocation failure in vector()\n");
                    return -1;
                }
                printf("Vector: ");
                for (i = 0; i < n; i++) {
                    vector[i] = (float) rand() / RAND_MAX;
                }
                break;
            }
            case 'f': {
                float *matrix;
                int m, n;
                printf("Allocate a float matrix with range [%d..%d] x [%d..%d]\n", 0, m-1, 0, n-1);
                matrix = (float **) malloc(sizeof(float *) * m);
                if (!matrix) {
                    printf("Allocation failure 1 in matrix()\n");
                    return -1;
                }
                for (i = 0; i < m; i++) {
                    matrix[i] = (float *) malloc(sizeof(float) * n);
                    if (!matrix[i]) {
                        printf("Allocation failure 2 in matrix()\n");
                        return -1;
                    }
                }
                break;
            }
            case 'd': {
                double *double_vector;
                int n;
                printf("Allocate a double vector with range [%d..%d]\n", 0, n-1);
                double_vector = (double *) malloc(sizeof(double) * n);
                if (!double_vector) {
                    printf("Allocation failure in double_vector()\n");
                    return -1;
                }
                printf("Double Vector: ");
                for (i = 0; i < n; i++) {
                    double_vector[i] = (double) rand() / RAND_MAX;
                }
                break;
            }
            case 'm': {
                unsigned char **matrix;
                int m, n;
                printf("Allocate a matrix with range [%d..%d] x [%d..%d]\n", 0, m-1, 0, n-1);
                matrix = (unsigned char **) malloc(sizeof(unsigned char *) * m);
                if (!matrix) {
                    printf("Allocation failure 1 in matrix()\n");
                    return -1;
                }
                for (i = 0; i < m; i++) {
                    matrix[i] = (unsigned char *) malloc(sizeof(unsigned char) * n);
                    if (!matrix[i]) {
                        printf("Allocation failure 2 in matrix()\n");
                        return -1;
                    }
                }
                break;
            }
            case 'n': {
                unsigned char *new_char;
                int n;
                printf("Allocate a character matrix with range [%d..%d] x [%d..%d]\n", 0, n-1, 0, n-1);
                new_char = (unsigned char *) malloc(sizeof(unsigned char) * n);
                if (!new_char) {
                    printf("Allocation failure in new_char()\n");
                    return -1;
                }
                printf("New Char: ");
                for (i = 0; i < n; i++) {
                    new_char[i] = (char) rand() / RAND_MAX;
                }
                break;
            }
            case 'r': {
                unsigned char *realloc_char;
                int n;
                printf("Allocate a character matrix with range [%d..%d] x [%d..%d]\n", 0, n-1, 0, n-1);
                realloc_char = (unsigned char *) malloc(sizeof(unsigned char) * n);
                if (!realloc_char) {
                    printf("Allocation failure in realloc_char()\n");
                    return -1;
                }
                printf("Realloc Char: ");
                for (i = 0; i < n; i++) {
                    realloc_char[i] = (char) rand() / RAND_MAX;
                }
                break;
            }
            case 's': {
                float *sizeof_float;
                int n;
                printf("Allocate a float vector with range [%d..%d]\n", 0, n-1);
                sizeof_float = (float *) malloc(sizeof(float) * n);
                if (!sizeof_float) {
                    printf("Allocation failure in sizeof_float()\n");
                    return -1;
                }
                printf("Sizeof Float: ");
                for (i = 0; i < n; i++) {
                    sizeof_float[i] = (float) rand() / RAND_MAX;
                }
                break;
            }
            case 't': {
                float *test_float;
                int n;
                printf("Allocate a float vector with range [%d..%d]\n", 0, n-1);
                test_float = (float *) malloc(sizeof(float) * n);
                if (!test_float) {
                    printf("Allocation failure in test_float()\n");
                    return -1;
                }
                printf("Test Float: ");
                for (i = 0; i < n; i++) {
                    test_float[i] = (float) rand() / RAND_MAX;
                }
                break;
            }
            case 'u': {
                float *unset_float;
                int n;
                printf("Allocate a float vector with range [%d..%d]\n", 0, n-1);
                unset_float = (float *) malloc(sizeof(float) * n);
                if (!unset_float) {
                    printf("Allocation failure in unset_float()\n");
                    return -1;
                }
                printf("Unset Float: ");
                for (i = 0; i < n; i++) {
                    unset_float[i] = (float) rand() / RAND_MAX;
                }
                break;
            }
            case 'w': {
                float *write_float;
                int n;
                printf("Allocate a float vector with range [%d..%d]\n", 0, n-1);
                write_float = (float *) malloc(sizeof(float) * n);
                if (!write_float) {
                    printf("Allocation failure in write_float()\n");
                    return -1;
                }
                printf("Write Float: ");
                for (i = 0; i < n; i++) {
                    write_float[i] = (float) rand() / RAND_MAX;
                }
                break;
            }
            case 'x': {
                float *exit_float;
                int n;
                printf("Allocate a float vector with range [%d..%d]\n", 0, n-1);
                exit_float = (float *) malloc(sizeof(float) * n);
                if (!exit_float) {
                    printf("Allocation failure in exit_float()\n");
                    return -1;
                }
                printf("Exit Float: ");
                for (i = 0; i < n; i++) {
                    exit_float[i] = (float) rand() / RAND_MAX;
                }
                break;
            }
        }
    }
}

```

【図64】

```

/* deallocate an char matrix */
void free_matrix(char **m, nrl, nrh, ncl, nch)
{
    int nrl, nrh, ncl, nch;

    for (i = nrh; i >= nrl; i--) free((char*) (m[i]+ncl));
    free((char**) (m+nrl));
}

/* deallocate a double matrix */
void free_matrix(double **m, nrl, nrh, ncl, nch)
{
    int nrl, nrh, ncl, nch;

    for (i = nrh; i >= nrl; i--) free((double*) (m[i]+ncl));
    free((double**) (m+nrl));
}

/* deallocate a float matrix */
void free_matrix(float **m, nrl, nrh, ncl, nch)
{
    int nrl, nrh, ncl, nch;

    for (i = nrh; i >= nrl; i--) free((float*) (m[i]+ncl));
    free((float**) (m+nrl));
}

/* deallocate an int matrix */
void free_matrix(int **m, nrl, nrh, ncl, nch)
{
    int nrl, nrh, ncl, nch;

    for (i = nrh; i >= nrl; i--) free((int*) (m[i]+ncl));
    free((int**) (m+nrl));
}

```

—267—

```

...../int nl, nh)
.....
.....struct block_rect *v;
.....
.....v = (struct block_rect *) malloc(sizeof(nh-nl));
.....
.....if (!v){
.....    printf("allocation error: new_block\n");
.....    exit(1);
.....}
.....
.....for ( i = nl; i <= nh; i++) {
.....    v[i].previous = v[i].next = NULL;
.....    v[i].currentsubblock = v[i].firstsubblock = NULL;
.....    v[i].firstontextblock = v[i].currenttextblock = NULL;
.....    v[i].in_group = 0;
.....    v[i].index = 0;
.....    v[i].numsubblock = 0;
.....    v[i].on_point = 0;
.....    v[i].pairleft = v[i].pairright = NULL;
.....    v[i].isorted = 0;
.....    v[i].idensity = 0;
.....    v[i].ingroup = 0;
.....    v[i].indensity = 0.0;
.....}
.....return v;
.....
.....Chain the block
.....
.....struct block_rect *new_rectblock(headblock)
.....struct block_rect *headblock;
.....{
.....    struct block_rect *n;
.....    n = new_block(0,0);
.....    if (headblock->firstsubblock == NULL)
.....        headblock->firstsubblock = n;
.....    else{
.....        headblock->currentsubblock->next = n;
.....        n->previous = headblock->currentsubblock;
.....        headblock->currentsubblock = n;
.....        headblock->numsubblock++;
.....    }
.....    return n;
.....}
.....
.....Allocate memory for Whiteblock
.....
.....struct whiteblock *new_whitebl, nb)
.....int nl, nh)
.....{
.....    struct whiteblock *v;
.....    int i;
.....    v = (struct whiteblock *) malloc(sizeof(nh-nl+1));
.....    if (!v){
.....        printf("allocation error: whiteblock nl = %d nh = %d\n", nl, nh);
.....    }
.....}

```

【図66】

```

        struct data_outline *new_data_outline(n1, n2)
        int n1, n2;
    {
        int i;
        struct data_outline *v;
        v = (struct data_outline *) malloc((unsigned)(n2-n1+1)*
            sizeof(struct data_outline));
        if (!v) {
            printf("allocation error: data_outline\n");
            exit(1);
        }
        for (i = n1; i < n2; i++) {
            v[i].total_left = v[i].total_right = 0;
            v[i].current_left = v[i].current_right = NULL;
            v[i].headleft = v[i].headright = NULL;
        }
        return v;
    }

    /*..... Allocate memory for h_outline
    /*.....
    struct h_outline *new_h_outline(target)
    int target;
    {
        struct h_outline *v;
        v = (struct h_outline *) malloc(sizeof(struct h_outline));
        if (!v) {
            printf("allocation error: h_outline\n");
            exit(1);
        }
        v->x = target;
        v->y = NULL;
        return v;
    }

    /*..... Search the existence of surrounding "one"
    /*.....
    int test_one_branch(x, y, newdirect, x1, y1, x2, y2)
    unsigned char *v;
    int x, y, newdirect;
    int x1, y1, x2, y2;
    {
        int k, j, newx, newy;
        k = search_map(newdirect, search_map(newdirect)[0]);
        for (j = 0; j < k; j++) {
            newx = search_map_center[j].x;
            newy = search_map_center[j].y;
            if ((p[newx][newy] == ON) ||
                (newx == x1 || (newx == x2) || (newy == y1 || (newy == y2))))
                return 1;
        }
        return 0;
    }

    /*..... Adding one element to data_outline on right side
    /*.....
    void free_pair(pair1, pair2, n1, n2)
    int n1, n2;
    {
        int i;
        if (pair1 == NULL) return;
        for (i = n1; i < n2; i++) {
            free((int *)pair1[i]);
            free((int *)pair2[i]);
        }
        free((int *)pair1[n1]);
        free((int *)pair2[n1]);
        pair1 = pair2 + 1;
    }

    /*..... Free pair in block
    /*.....
    void free_block(pair1, pair2, n1, n2)
    int n1, n2;
    {
        int i;
        if (pair1 == NULL) return;
        for (i = n1; i < n2; i++) {
            free((int *)pair1[i]);
            free((int *)pair2[i]);
        }
        free((int *)pair1[n1]);
        free((int *)pair2[n1]);
        pair1 = pair2 + 1;
    }

    /*..... Free white
    /*.....
    void free_white(white)
    struct whiteblock *white;
    {
        free(pair1[white->pairright], white->pairleft, white->supper_y, white->lower_y);
        free(pair2[white->pairright], white->pairleft, white->supper_y, white->lower_y);
        white->whiteblock = NULL;
    }

    /*..... Allocate memory for data_outline
    /*.....
    struct data_outline *new_data_outline(n1, n2)
    int n1, n2;
    {
        int i;
        struct data_outline *v;
        v = (struct data_outline *) malloc((unsigned)(n2-n1+1)*
            sizeof(struct data_outline));
        if (!v) {
            printf("allocation error: data_outline\n");
            exit(1);
        }
        for (i = n1; i < n2; i++) {
            v[i].total_left = v[i].total_right = 0;
            v[i].current_left = v[i].current_right = NULL;
            v[i].headleft = v[i].headright = NULL;
        }
        return v;
    }

```


【図67】

```

.....
void add_x_right(x, y, ylist)
{
    int x;
    struct data_outline *ylist;
    {
        struct data_outline *top;
        if (ylist[y].total_right == 0)
        {
            ylist[y].total_right = top;
            ylist[y].headright = top;
        }
        else
        {
            ylist[y].current_right->next = top;
            ylist[y].current_right = top;
        }
        ylist[y].total_right++;
    }
}

.....
..... Adding one element to data_outline on left side
.....
void add_x_left(x, y, ylist)
{
    struct data_outline *ylist;
    {
        struct data_outline *top;
        top = new_data_outline(x);
        if (ylist[y].total_left == 0)
        {
            ylist[y].current_left = top;
            ylist[y].headleft = top;
        }
        else
        {
            ylist[y].current_left->next = top;
            ylist[y].current_left = top;
        }
        ylist[y].total_left++;
    }
}

.....
..... Adding element to data_outline
.....
void add_y_list(y, location, ylist)
{
    int location;
    struct data_outline *ylist;
    {
        if (location == LEFT)
        {
            add_x_left(x, y, ylist);
        }
        else if (location == RIGHT)
        {
            add_x_right(x, y, ylist);
        }
    }
}

.....
..... Sort integer array from index nl to nh
.....
void bubble_sort(int, nl, nh)
{
    int nl, nh;
}

```

【図68】

```

void set_outline(x, y, nblock)
int x, y;
{
    struct whiteblock *nblock;

    if (x < pairright[0]) {
        capair[numpair-2] = pairleft[0];
        capair[numpair-1] = pairright[0];
        for (i = 0; i < numpair; i++) {
            if (i < pairright[0]) {
                pairleft[i] = pairleft[i];
                pairright[i] = pairright[i];
            } else {
                pairleft[i] = pairleft[i];
                pairright[i] = pairright[i];
            }
        }
        numpair++;
    } else {
        if (pairright[0] < pairleft[0]) {
            capair[numpair-2] = pairright[0];
            capair[numpair-1] = pairleft[0];
            break;
        } else {
            pairright[0] = pairleft[0];
            pairleft[0] = pairright[0];
        }
    }
    free((int *)capair);

    pairright[0] = pairleft[0];
    for (i = 1; i < numpair; i++) {
        pairleft[i] = capair[i-1];
        pairright[i] = capair[i];
    }
    free((int *)capair);

    /* Search the direction of next 'one' */
    void rightmost_and(p, old_direct, i, j, x1, y1, x2, y2, new_direct, new_x, new_y)
    unsigned char *p;
    int i, j, x1, y1, x2, y2;
    int old_direct;
    int new_direct;
    int new_x;
    int new_y;

    for (i = 1; i < numpair; i++) {
        if (i < pairright[0]) {
            pairleft[i] = pairleft[i];
            pairright[i] = pairright[i];
        } else {
            pairleft[i] = pairleft[i];
            pairright[i] = pairright[i];
        }
    }
    numpair++;

    if (pairright[0] < pairleft[0]) {
        capair[numpair-2] = pairright[0];
        capair[numpair-1] = pairleft[0];
        break;
    } else {
        pairright[0] = pairleft[0];
        pairleft[0] = pairright[0];
    }
    free((int *)capair);

    pairright[0] = pairleft[0];
    for (i = 1; i < numpair; i++) {
        pairleft[i] = capair[i-1];
        pairright[i] = capair[i];
    }
    free((int *)capair);

    /* Set up the extreme point for the searched block */
    void set_outline(x, y, nblock)
    int x, y;
    struct whiteblock *nblock;

    if (x > nblock->right_x) {
        nblock->right_x = x;
    } else if (x < nblock->left_x) {
        nblock->left_x = x;
    }

    if (y > nblock->lower_y) {
        nblock->lower_y = y;
    } else if (y < nblock->upper_y) {
        nblock->upper_y = y;
    }

    /* Set up the extreme point for the searched block */
    void set_outline(x, y, nblock)
    int x, y;
    struct whiteblock *nblock;

    if (x > nblock->right_x) {
        nblock->right_x = x;
    } else if (x < nblock->left_x) {
        nblock->left_x = x;
    }

    if (y > nblock->lower_y) {
        nblock->lower_y = y;
    } else if (y < nblock->upper_y) {
        nblock->upper_y = y;
    }

    /* Allocate int* memory */
    int *nblock;
    int *nblock;

    int *nblock;
    int *nblock;

    if (y < (int *)nblock) {
        printf("allocation error in intblock", y);
        exit(1);
    }
    y = nblock;
    return y;

    /* Transfer outline data into pair */
    void put_outline_into_pair(nblock, ylist)
    struct whiteblock *nblock;
    struct data_outline ylist;

    int i, k;
    struct data_outline *capair;

    nblock->pairright = (int *)nblock;
    nblock->pairleft = (int *)nblock;
    for (i = nblock->upper_y; i < nblock->lower_y; i++) {

```

【図69】

```

n-blank = 1;
n-supply = first-supply;
n-lower = first-lower;
n-right = first-right;
n-left = first-left;
n-y = first-y;
n-x = first-x;
n-width = first-width;
n-length = first-length;
n-on-point = first-on-point;
n-first = first-first;
n-sorted = first-sorted;
n-pairleft = first-pairleft;
n-pairright = first-pairright;
first-pairleft = NULL;
first-pairright = NULL;
n-order = ORIGINAL;

first-subblock = 1;
first-firstsubblock = n;
first-currentsubblock = NULL;
first-nextsubblock = NULL;
/* print out of conversion */

/*----- Add one block under the specified block by the y location -----*/
void add_rectblock(text, first, head)
int text;
struct block *first;
struct block *head;
{
    struct block *new;
    if (first-supply < head-supply) {
        head-supply = first-supply;
        head-length = head-slower-head-supply;
    }
    if (first-lower > head-lower) {
        head-lower = first-lower;
        head-width = head-slower-head-supply;
    }
    if (first-left < head-left) {
        head-left = first-left;
        head-right = head-right;
    }
    if (first-right > head-right) {
        head-right = first-right;
        head-width = head-right-head-left;
    }
    if (text != TEXT) {
        if (head-firstsubblock == NULL) {
            head-firstsubblock = first;
            first-previous = first-slow;
            head-currentsubblock = first;
        }
        else {
            for (tmp = head-currentsubblock; tmp->next != NULL; tmp = tmp->next) {
                if (first-previous == NULL) {
                    first-next = head-firstsubblock;
                    head-firstsubblock->previous = first;
                }
            }
        }
    }
}

nblock-pairleft[i] = (int *)vector(0, ylist[i].total_right);
nblock-pairright[i] = (int *)vector(0, ylist[i].total_left);
nblock-pairleft[i][0] = ylist[i].total_right;
nblock-pairright[i][0] = ylist[i].total_left;
for (tmp1 = ylist[i].headright, k = 1; tmp1 != NULL; k++) {
    nblock-pairleft[i][k] = tmp1->x;
    nblock-pairright[i][k] = tmp1->y;
    free(struct h_outline *tmp1);
    tmp1 = tmp2;
}
nblock-pairleft[i][0] = ylist[i].total_left;
for (tmp1 = ylist[i].headleft, k = 1; tmp1 != NULL; k++) {
    nblock-pairright[i][k] = tmp1->x;
    nblock-pairleft[i][k] = tmp1->y;
    free(struct h_outline *tmp1);
    tmp1 = tmp2;
}

/*----- Transfer outline data into pair -----*/
void write_outline_into_pair(nblock, ylist)
struct block *nblock;
struct data_outline ylist[];
{
    int i, k;
    struct h_outline *tmp1, *tmp2;

    nblock-pairright = (int **)vector(nblock->upper_y, nblock->lower_y);
    nblock-pairleft = (int **)vector(nblock->upper_y, nblock->lower_y);
    for (i = nblock->upper_y; i < nblock->lower_y; i++) {
        nblock-pairright[i] = (int *)vector(0, ylist[i].total_right);
        nblock-pairleft[i] = (int *)vector(0, ylist[i].total_left);
        for (tmp1 = ylist[i].headright, k = 1; tmp1 != NULL; k++) {
            nblock-pairright[i][k] = tmp1->x;
            nblock-pairleft[i][k] = tmp1->y;
            free(struct h_outline *tmp1);
            tmp1 = tmp2;
        }
        nblock-pairright[i][0] = ylist[i].total_right;
        for (tmp1 = ylist[i].headleft, k = 1; tmp1 != NULL; k++) {
            nblock-pairleft[i][k] = tmp1->x;
            nblock-pairright[i][k] = tmp1->y;
            free(struct h_outline *tmp1);
            tmp1 = tmp2;
        }
        if (first-firstsubblock == NULL) {
            first-firstsubblock = nblock;
            first-previous = first-slow;
            first-currentsubblock = first;
        }
    }
}

```

【図70】

```

head->firstnonemptyblock = first;
break;
} else if (first->upper_y < temp->upper_y) continue;
else {
    first->previous = temp;
    first->next = temp->next;
    temp->next = first;
    if (first->next != NULL)
        head->currentnonemptyblock = first;
    else
        head->currentnonemptyblock = first;
    break;
}
} else {
    if (head->firstsubblock == NULL) {
        head->firstsubblock = first;
        first->previous = first->next = NULL;
        head->currentsubblock = first;
    } else {
        for (temp = head->currentsubblock; temp->next != NULL; temp = temp->next) {
            first->previous = NULL;
            first->next = head->firstsubblock->previous;
            head->firstsubblock->previous = first;
            break;
        }
        else if (first->upper_y < temp->upper_y) continue;
        else {
            first->previous = temp;
            first->next = temp->next;
            temp->next = first;
            if (first->next != NULL)
                head->currentsubblock = first;
            else
                head->currentsubblock = first;
            break;
        }
    }
}

void remove_block(struct first, head)
{
    struct block *first, *head;
    if (head->firstsubblock == first) {
        head->firstsubblock = first->next;
        if (first->next != NULL)
            head->currentsubblock = first->next;
        else
            head->currentsubblock = NULL;
    } else if (head->currentsubblock == first) {
        head->currentsubblock = first->previous;
        if (first->previous != NULL)
            first->previous->next = first->previous;
        else
            first->previous = first->previous;
    }
}

void remove_nonempty_block(struct first, head)
{
    struct block *first, *head;
    if (head->firstnonemptyblock == first) {
        head->firstnonemptyblock = first->next;
        if (first->next != NULL)
            head->currentnonemptyblock = first->next;
        else
            head->currentnonemptyblock = NULL;
    } else if (head->currentnonemptyblock == first) {
        head->currentnonemptyblock = first->previous;
        if (first->previous != NULL)
            first->previous->next = first->previous;
        else
            first->previous = first->previous;
    }
}

void transfer_block(struct first, totat, first, originalhead, head)
{
    struct block *first, *originalhead, *head;
    /* print: go into transfer block */
    if (first->next == NULL)
        remove_block(first, originalhead);
    else
        remove_block(first, originalhead);
    add_block(totat, first, head);
    if (originalhead != head) {
        originalhead->nextsubblock--;
    }
}

```

【圖 7 1】

```

        head_vnumsubblock++)
    }
    printf("get out of transfer block\n");
}

/*..... Rearrange the nested relation among the block
.....*/
void rearrange_rectblock(headblock)
struct block_rect *headblock;
{
    struct block_rect *tap, *tappb, *tp;
    /* printf("get into rearrange_rectblock\n"); */
    tap = headblock->firstsubblock;
    for ( ; tap != NULL; tap = tappb)
    {
        tappb = tap->next;
        for (tapp = tap->next; tapp != NULL; tapp = tapp->next)
        {
            if ((tapp->lower_y > tap->lower_y) ||
                (tapp->left_x < tap->left_x) ||
                (tapp->right_x < tap->right_x) ||
                (tapp->upper_y < tap->upper_y))
            {
                tapp->cover = COVER;
                tapp = tapp->previous;
                tapp->cover = COVERED;
                transfer_block(TEXT, TEXT, tapp, headblock, tapp);
                if (tappb == tapp) tappb = tapp->next;
            }
            else if ((tapp->lower_y == tap->lower_y) ||
                    (tapp->left_x == tap->left_x) ||
                    (tapp->right_x == tap->right_x) ||
                    (tapp->upper_y == tap->upper_y))
            {
                converted_headblock(tapp);
                if (tapp->cover == COVER)
                {
                    for (b = tap->firstsubblock; b != NULL;
                        b = b->next)
                    {
                        transfer_block(TEXT, TEXT, b, tapp, tapp);
                        b->cover = COVERED;
                    }
                }
                else
                {
                    remove_rectblock(tapp, headblock);
                    tapp->cover = COVERED;
                    transfer_block(TEXT, TEXT, tapp,
                        headblock, tapp);
                }
            }
            break;
        }
    }
}

/*..... PRINT out of rearrange_rectblock\n */
/*..... SEARCH & CONNECTED 'one' component
.....*/
void search_one(p, startx, starty, x1, y1, x2, y2, nblock)
{
    struct block_rect *nblock;
    int oldx, oldy, newx, newy;
    int olddirect, newdirect;
    struct data_outline *ylist_on;
    int i, j;

    /* printf("get into search one x = %d y = %d\n", startx, starty); */
    ylist_on = (struct data_outline *) new_data_outline(y1, y2);
    nblock->nx = startx;
    nblock->ny = starty;
    nblock->left_x = nblock->right_x = startx;
    nblock->upper_y = nblock->lower_y = starty;
    if ((list_one_branchup, startx, starty, EAST, x1, y1, x2, y2) == 0)
    {
        nblock->on_density = 1.0;
        nblock->on_density = 1.0;
        nblock->on_density = 1.0;
        pstartx[startx] = LEFTRIGHT_ONE;
        free((struct data_outline *) (ylist_on->ny));
        return;
    }
    for (i = startx-2; i <= startx+20; i++)
    {
        printf("old ", i);
        for (j = startx-10; j <= startx+10; j++)
        {
            printf("new ", p[i][j]);
        }
        printf("\n");
    }
    if (p[startx][startx+1] == 0)
    {
        pstartx[startx] = RIGHT_ONE;
        sdy_list(startx, startx, x1, ylist_on);
        pstartx[startx] = LEFTRIGHT_ONE;
    }
    else
    {
        sdy_list(startx, startx, x1, ylist_on);
        olddirect = RIGHT;
        right_one_one(p, olddirect, startx, starty, x1, y1, x2, y2);
        set_outline_in_one(p, olddirect, startx, starty, x1, y1, x2, y2);
        oldx = newx, oldy = newy;
        olddirect = newdirect;
        for ( ; )
        {
            right_one_one(p, olddirect, oldx, oldy, x1, y1, x2, y1);
            newdirect = newx, newy, oldirection;
            if (location != STOP)
            {
                sdy_list(oldx, oldy, location, ylist_on);
                if ((location == STOP) || (oldx == oldy))
                {
                    poldy[oldx] = LEFTRIGHT_ONE;
                    poldy[oldx] = LEFT_ONE;
                }
                if (location == RIGHT)
                {
                    if (poldy[oldx] != LEFT_ONE)
                    {
                        poldy[oldx] = RIGHT_ONE;
                    }
                }
            }
        }
    }
}

```

【图 7 2】

—274—

—275—

```
(free(struct_data_outline *) (y1list_of_block->upper_y1);
return while;
```

[illegible]

—277—

—277—

—278—

—278—

【图 7 7】

[illegible]

—280—

```

twp->act = UNKNOWN;
if (twp->cover == COVER) {
    for (twp = twp->firstsubblock; twp != NULL; twp = twp->next) {
        if (twp->cover == ORIGINAL) {
            transfer_block(TEXT, NOWTEXT, twp, twp, twp);
        }
    }
}

twp = twp->next;
transfer_block(TEXT, NOWTEXT, twp, headblock, headblock);
twp = twp;

else {
    twp->length += (long)twp->length;
    twp->act = TEXT;
    twp = twp->next;
}

if (num > 0)
    headblock->avg_height = (float)twp->length/(float)num;
else
    headblock->avg_height = 0;
return num;
}

if (twp->act == TEXT)
    printf("get out of least_discriminate\n");
}

void block_split(head, twp, gpt, gpt)
struct block *twp, *head;
int gpt, gpt;
{
    struct block *rect = headblock, *b, *twp;
    int i, y;

    printf("last time block split = %d y = %d cover = %d first = %d firstsubblock = %d\n", twp->act, twp->cover, twp->firstsubblock, twp->firstsubblock);

    headblock = new_block(0,0);
    b = twp->firstsubblock;

    headblock->pairleft = b->pairright;
    headblock->pairleft = b->pairleft;
    headblock->supper_y = gpt;
    headblock->right_x = twp->right_x;
    headblock->right_x = b->right_x;
    headblock->left_x = b->left_x;
    for (y = headblock->supper_y; y == headblock->lower_y; y++)
        for (i = 1; i < b->pairleft(y)[0]; i++)
            if (headblock->right_x < b->pairleft(y)[1])
                headblock->right_x = b->pairleft(y)[1];
    for (y = headblock->supper_y; y == headblock->lower_y; y++)
        for (i = 1; i < b->pairright(y)[0]; i++)
            if (headblock->left_x > b->pairright(y)[1])
                headblock->left_x = b->pairright(y)[1];

    headblock->length = headblock->lower_y - headblock->supper_y + 1;
    headblock->width = headblock->right_x - headblock->left_x + 1;
    headblock->act = gpt;
    headblock->next = twp->next;
    headblock->length = twp->length;
}

if ((headblock->act == PAUSE) || (headblock->act == STOP)) {
    printf("before search block in block\n");
    search_block_in_block(headblock, p);
}

printf("after search block in block\n");
if ((headblock->firstsubblock != NULL) && (headblock->group == 2))
    sort_block(headblock);

for (twp = headblock->firstsubblock; twp != NULL; twp = twp->next)
    printf("index = %d in group = %d\n", twp->index, twp->in_group);

rearrange_rectblock(headblock);

for (twp = headblock->firstsubblock; twp != NULL; twp = twp->next)
    printf("index = %d in group = %d\n", twp->index, twp->in_group);

if ((headblock->pairright != NULL) && (headblock->pairleft != NULL) && (headblock->supper_y != headblock->lower_y))
    return;

printf("act = %d\n", headblock->act);
printf("y = %d\n", headblock->supper_y);

return;

printf("get out of block in block\n");
}

void delete_block_from_the_head_block()
{
    struct block *rect = first_block, *headblock;

    remove_rectblock(first_block);
    headblock->nextsubblock--;
    free_rectblock(first_block);
}

int least_discriminate(headblock)
struct block *rect = headblock;

int num = 0;
long twp->length = 0;
float twp->act;
struct block *twp, *twp;

printf("get into least_discriminate\n");
twp->length = (float)1000000000;
for (twp = headblock->firstsubblock; twp != NULL; twp = twp->next)
    if ((twp->act == PAUSE) || (twp->act == STOP) && headblock->width > twp->length)
        twp->length = twp->length;
}

```

【図7.9】

```

    struct block *next, *b;
    int i, y;

    printf("get into block split x = %d y = %d cover = %d first = %d last = %d\n",
           tap->left_x, tap->upper_y, tap->cover, tap->firstsubblock,
           tap->firstsubblock);

    newblock = new_block(0, 0);
    newblock->splitright = tap->splitright;
    newblock->splitleft = tap->splitleft;
    newblock->upper_y = gap;
    newblock->lower_y = tap->lower_y;

    newblock->right_x = tap->right_x;
    newblock->left_x = tap->left_x;
    for (y = newblock->upper_y; y < newblock->lower_y; y++)
        for (i = 1; i < tap->splitright[y][0]; i++)
            if (newblock->right_x < tap->splitright[y][i])
                newblock->right_x = tap->splitright[y][i];
    for (y = newblock->upper_y; y < newblock->lower_y; y++)
        for (i = 1; i < tap->splitleft[y][0]; i++)
            if (newblock->left_x > tap->splitleft[y][i])
                newblock->left_x = tap->splitleft[y][i];

    newblock->length = newblock->lower_y - newblock->upper_y + 1;
    newblock->width = newblock->right_x - newblock->left_x + 1;
    newblock->on_point = tap->on_point + newblock->length / tap->length;

    newblock->y = gap;

    newblock->x = newblock->right_x;
    for (i = 1; i < tap->splitright[newblock->upper_y][0]; i++)
        if (tap->splitright[newblock->upper_y][i] < newblock->x)
            newblock->x = tap->splitright[newblock->upper_y][i];

    newblock->in_group = tap->in_group;
    newblock->cover = tap->cover;

    tap->lower_y = gap;
    tap->length = tap->lower_y - tap->upper_y + 1;
    tap->on_point = tap->on_point - newblock->on_point;

    add_rectblock(TEXT, newblock, head);

    printf("newblock->previous->index = %d\n", newblock->previous->index);

    for (b = b->next; b != NULL; b++)
        printf("b = %d b-y = %d b-y1 = %d tap->first = %d y1 = %d y2 = %d\n",
               b->index, b->upper_y, b->lower_y, tap->firstsubblock->index,
               tap->firstsubblock->upper_y, tap->firstsubblock->lower_y);

    if (lower_range(b->upper_y, b->lower_y, tap->firstsubblock->upper_y,
                    tap->firstsubblock->lower_y) < 0.5) {
        tap = b->next;
        newblock->length = newblock->length;
        newblock->cover = COVER;
        transfer_block(TEXT, TEXT, b, tap, newblock);
        b = tap;
    } else b = b->next;

    if (tap->firstsubblock->next == NULL) {
        tap->firstsubblock->index = tap->index;
        transfer_block(TEXT, TEXT, tap->firstsubblock, tap, head);
        remove_textblock(tap, head);
    } else {
        tap->upper_y = tap->firstsubblock->upper_y;
        tap->lower_y = tap->firstsubblock->lower_y;
        tap->left_x = tap->firstsubblock->left_x;
        tap->right_x = tap->firstsubblock->right_x;
        for (i = tap->upper_y; i < tap->lower_y; i++)
            if (tap->upper_y > b->upper_y)
                tap->upper_y = b->upper_y;
            if (tap->lower_y < b->lower_y)
                tap->lower_y = b->lower_y;
            if (tap->left_x > b->left_x)
                tap->left_x = b->left_x;
            if (tap->right_x < b->right_x)
                tap->right_x = b->right_x;
        tap->width = tap->right_x - tap->left_x + 1;
        tap->length = tap->lower_y - tap->upper_y + 1;
    }

    printf("get out of block split\n");

    // ..... split the long connected component if some gap between .....
    // .....
    int text_split(headblock, tap, len)
    struct block *next, *b;
    {
        int i, j, total, split;
        int upper, lower, left, right;
        int gap, gap2;

        printf("test into text split: len = %d\n", len);
    }

```

【図80】

```

print("index = %d y= %d x= %d len = %d tap= %d\n",
      tap->index, tap->upper_y, tap->left_x, len, tap->length);
//
split = 0;
if (((float)tap->length > len*TEXT_SCALE))
    if ((float)tap->length < len*TEXT_SCALE*2)
    {
        upper = (tap->upper_y-1)*reduce_factor;
        lower = (tap->upper_y-1)*reduce_factor;
        left = (tap->left_x-1)*reduce_factor;
        right = (tap->right_x-1)*reduce_factor;
    }
    print("place %d\n",
          total_xvector(upper, lower);
    for (i = 0; i <= lower; i++)
        total[i] = 0;
    for (j = left; j <= right; j++)
        if (original_p[i][j]>0) total[i]++;
    /* print("total = %d\n", total[0]); */
    for (i = upper; total[i] <= 0; i++)
        if (total[i] <= 0)
        {
            for (gap1 = 1; gap1 <= total[i]; gap1++)
            {
                for (gap2 = 1; gap2 <= total[i]; gap2++)
                {
                    if (gap1 == lower) break;
                    gap1 = (gap1+random_factor)/2;
                    if (gap1 == gap2)
                    {
                        if (gap1 == tap->upper_y) gap1 = tap->upper_y+1;
                        else if (gap1 == tap->lower_y) gap1 = tap->lower_y-1;
                        else gap1 = gap1+1;
                    }
                }
            }
        }
    /* print("gap1 = %d gap2 = %d x = %d y = %d\n", gap1, gap2, tap->left_x,
          tap->upper_y); */
    if (tap->firstsubblock != NULL)
        block_split(headblock, tap, gap1, gap2);
    else
        block_split(headblock, tap, gap1, gap2);
    split = 1;
    break;
}
free_vector(total, upper, lower);
return split;
/* print("get out of text_split\n"); */
//
//----- Second time text discrimination -----//
//----- Count the density inside the block -----//
void s_text_discriminate(headblock)
{
    struct block_rect *headblock;
    int num = 0;
    float temp_length = 0;
    struct block_rect *tap, *tempb;
}

```

【図81】

```

void block_density(p, block)
unsigned char **p;
struct block_rect *block;
{
    int i, j, k, zero_point;
    printf("get into block_density\n");
    // sort_for_block(block);
    for (i = block->upper_y; i <= block->lower_y; i++)
    {
        for (j = block->left_x; j <= block->right_x; j++)
        {
            if (p[i][j] == 0) printf("0");
            if (p[i][j] == 255) printf("255");
            if (p[i][j] == 127) printf("127");
            if (p[i][j] == 128) printf("128");
            if (p[i][j] == 129) printf("129");
        }
        printf("\n");
    }

    for (i = zero_point; i <= block->upper_y; i++)
    {
        for (j = 1; j <= block->right_x[0]; j++)
        {
            if (block->right_x[i][j] <= block->right_x[i][0]) k++;
            else if (p[i][j] == 0) block->zero_point++;
        }
        printf("zero_point = %d\n", block->zero_point);
        block->density = (float)(block->zero_point)/(float)(block->right_x->length);
    }

    // Horizontal solid line testing
    // Vertical solid line testing
    // =====
    int i, j;
    struct block_rect *block;
    unsigned char **p;
    struct block_rect *block_rect;
    float sigma;

    printf("place\n");
    histogram = (int *)vector(block->upper_y, block->right_x);
    for (i = 0; i <= block->upper_y; i++)
    {
        for (j = block->left_x; j <= block->right_x; j++)
        {
            if (p[i][j] == 0) histogram[i]++;
            if (p[i][j] == 255) histogram[i]++;
            if (p[i][j] == 127) histogram[i]++;
            if (p[i][j] == 128) histogram[i]++;
            if (p[i][j] == 129) histogram[i]++;
        }
        sigma += (float)(histogram[i])/(float)(block->width);
    }
    free((int *)histogram);
    sigma /= block->length;
    block->density(p, block);
    printf("block_density\n");
    if (sigma <= SOLIDITY)
    {
        return SOLID;
    }
    else
    {
        for (i = 0; i <= block->upper_y; i++)
        {
            for (j = block->left_x; j <= block->right_x; j++)
            {
                sigma += (float)(p[i][j]);
            }
            sigma /= block->length;
            if (sigma <= SOLIDITY)
            {
                return SOLID;
            }
            else
            {
                for (i = 0; i <= block->upper_y; i++)
                {
                    for (j = block->left_x; j <= block->right_x; j++)
                    {
                        sigma += (float)(p[i][j]);
                    }
                }
            }
        }
    }
}

```

【図82】

```

for (j = block->left_x; j <= block->right_x; j++)
{
    pti[j] = 0;
    sigma = ((float) (j - block->left_x)) /
        (float) (block->right_x - block->left_x);
    sigma /= block->length;
    if (sigma <= 0.01)
        return VERDUP;
    else
        return UNKNOWN;
}

/*----- Frame testing -----*/
void frame_discriminate(p, block)
unsigned char *p;
struct block *block;
{
    int num, w, l, longest, j, zaplong, ti;
    long sigma;

    /* Print "get into frame_dis" */
    /* sort_for_block(block);
    if ( (block->length > 150/SCALE_RATIO) || (block->width > 300/SCALE_RATIO) ) {
        c = 1;
        while ( (float) (block->length * 0.5 / SCALE_RATIO) > 0.5 ||
                (float) (block->width * 0.5 / SCALE_RATIO) > 0.5 ) {
            for (sum=0, i=block->upper_y-1; i <= block->lower_y-1; i++) {
                for (longest=0, j=1; j <= block->pairright[i][0]; j++) {
                    if ( (float) (block->pairright[i][j] - block->pairright[i][j-1]) > longest )
                        longest = block->pairright[i][j] - block->pairright[i][j-1];
                    sigma += (long) (block->width * longest);
                }
                num++;
            }
            if (num > 0) sigma /= num;

            /* frame test and content test */
            if ( (num > 1) || ((float) (block->length * 2) / 0.1) <= (sigma <= c * 2) )
                block->att = FRAME;
            else if ( ((float) block->length < 150/SCALE_RATIO) ||
                    ((float) block->width < 300/SCALE_RATIO) )
                block->att = if_block(p, block);
            else if ( ((float) block->width < 150/SCALE_RATIO) ||
                    ((float) block->length < 300/SCALE_RATIO) )
                block->att = if_vert(p, block);
        }
        printf("att = %d\n", block->att);
    }

    /*----- Recover the covered block into original head block -----*/
    void text_recover(head, first);
    struct block *head, *first;
}

for (j = block->left_x; j <= block->right_x; j++)
{
    pti[j] = 0;
    sigma = ((float) (j - block->left_x)) /
        (float) (block->right_x - block->left_x);
    sigma /= block->length;
    if (sigma <= 0.01)
        return VERDUP;
    else
        return UNKNOWN;
}

/*----- Frame testing -----*/
void frame_discriminate(p, block)
unsigned char *p;
struct block *block;
{
    int num, w, l, longest, j, zaplong, ti;
    long sigma;

    /* Print "get into frame_dis" */
    /* sort_for_block(block);
    if ( (block->length > 150/SCALE_RATIO) || (block->width > 300/SCALE_RATIO) ) {
        c = 1;
        while ( (float) (block->length * 0.5 / SCALE_RATIO) > 0.5 ||
                (float) (block->width * 0.5 / SCALE_RATIO) > 0.5 ) {
            for (sum=0, i=block->upper_y-1; i <= block->lower_y-1; i++) {
                for (longest=0, j=1; j <= block->pairright[i][0]; j++) {
                    if ( (float) (block->pairright[i][j] - block->pairright[i][j-1]) > longest )
                        longest = block->pairright[i][j] - block->pairright[i][j-1];
                    sigma += (long) (block->width * longest);
                }
                num++;
            }
            if (num > 0) sigma /= num;

            /* frame test and content test */
            if ( (num > 1) || ((float) (block->length * 2) / 0.1) <= (sigma <= c * 2) )
                block->att = FRAME;
            else if ( ((float) block->length < 150/SCALE_RATIO) ||
                    ((float) block->width < 300/SCALE_RATIO) )
                block->att = if_block(p, block);
            else if ( ((float) block->width < 150/SCALE_RATIO) ||
                    ((float) block->length < 300/SCALE_RATIO) )
                block->att = if_vert(p, block);
        }
        printf("att = %d\n", block->att);
    }

    /*----- Recover the covered block into original head block -----*/
    void text_recover(head, first);
    struct block *head, *first;
}

struct block *tap, *tapnext;
float
tapl = (head->avg_height == 0.0) ? text_length : head->avg_height;
for (tap = first->firstblock; tap != NULL; )
{
    if (tap->cover == COVERED)
        if ( ((float) tap->length <= tapl * TEXT_SCALE)
            tapnext = tap->next;
            tap = tapnext;
        else
            if (tap->length > tapl * TEXT_SCALE)
                if (text_split(first, tap, tapl)) continue;
                tapnext = tap->next;
                transfer_block(TEXT, NEWTEXT, tap, first, head);
                tap = tapnext;
            else
                tap = tap->next;
    }
    else
        tap = tap->next;
}

/*----- Recover the covered content block into original head block -----*/
void text_recover(head, first);
struct block *head, *first;

struct block *tap, *tapnext;
float
tapl = (head->avg_height == 0.0) ? text_length : head->avg_height;
for (tap = first->firstblock; tap != NULL; )
{
    if (tap->cover == COVERED)
        tapnext = tap->next;
        if ( ((float) tap->length <= tapl * TEXT_SCALE)
            tapnext = tap->next;
            tap = tapnext;
        else
            if (tap->length > tapl * TEXT_SCALE)
                if (text_split(first, tap, tapl)) continue;
                tapnext = tap->next;
                transfer_block(TEXT, NEWTEXT, tap, first, head);
                tap = tapnext;
            else
                tap = tap->next;
    }
    else
        tap = tap->next;
}

/*----- Processing for One content block continued -----*/
void one_content_contained(block, head)

```


【図83】

```

struct block_rect *block, *head;
{
    struct block_rect *tap;
    /* ..... Classify nontext block ..... */
    void nt_class(p, tapb);
    unsigned char *p;
    struct block_rect *tapb;
    {
        struct block_rect *subb;
        for (subb = tapb->firstnontextblock; subb != NULL; subb = subb->next)
            nt_class(subb, tapb);
    }
    /* ..... Mifflone testing ..... */
    int if_halfone(block, head);
    struct block_rect *block, *head;
    int half = 0;
    int num_tap;
    struct block_rect *tap, *tapb;
    /* ..... */
    /* printf "get into if_halfone, x = %d y = %d\n", block->left_x, block->upper_y, */
    if (block->firstnontextblock == NULL) return half;
    else {
        for (num_t = 0, dot = 0, tap = block->firstnontextblock;
             tap != NULL; tap = tap->next) {
            /* printf "inside %d %d y = %d x = %d\n", tap->upper_y, tap->length, tap->width, */
            tap->index, if (((float)tap->length < HPOINT/SCALE_RATIO) ||
                           ((float)tap->width < HPOINT/SCALE_RATIO))
                dot++;
            num_t++;
        }
        /* printf "dot = %d num = %d\n", dot, num_t, */
        if ((dot > HPOINT) && (dot > ((num_t+1)/2)))
            half = HPOINT/SCALE_RATIO;
        /* ..... */
        if (half < HPOINT/SCALE_RATIO) {
            for (tap = block->firstnontextblock; tap != NULL; )
                if (((float)tap->length < HPOINT/SCALE_RATIO) ||
                    ((float)tap->width < HPOINT/SCALE_RATIO))
                    if (tap->cover == COVER) {
                        tapb = tap->next;
                        transfer_block(TEXT, TEXT, tap, block, head);
                        tap = tapb;
                    } else tap = tap->next;
            start_block_rect(block, head);
            one_nontext_contained(block);
        }
    }
    /* ..... */
    /* printf "get out of one nontextblock\n", */
    /* ..... */
    /* printf "get out of one nontextblock\n", */
    /* ..... */
    void nt_class(p, tapb, head);
    unsigned char *p;
    struct block_rect *tapb, *head;
    struct block_rect *tapnub;
    /* printf "get into nt_class\n", */
    if (tap->firstnontextblock == NULL)
        return;
    for (tapnub = tap->firstnontextblock; tapnub != NULL;
         tapnub = tapnub->next) {
        frame_discriminate(p, tapnub);
        if (((tapnub->start == HPOINT/SCALE_RATIO) || (tapnub->end == (HPOINT/SCALE_RATIO)))
            && (tapnub->cover == COVER) && (tapnub->cover == COVER)) {
            text_recover(head, tapb);
            nt_class_recover(head, tapb);
            one_nontext_contained(tapb);
        }
    }
    if (frame_discriminate(p, tapb))
        /* printf "get out nontext_class\n", */
}

```

【図84】

```

print("get out of halfcos test\n");
return half;
}

.....
Discriminate cur inside nontext block
.....
void text_in_block(first)
struct block_rect *first;
{
    struct block_rect *tap;
    for (tap = first; tap != NULL; tap = tap->next) {
        if (tap->rect.picture == NULL) continue;
        if (tap->firstsubblock != NULL) text_discriminate(tap);
    }
    print("get out of text in nontext\n");
}

.....
void halfcos_test(testblock)
struct block_rect *testblock;
{
    struct block_rect *tap;
    for (tap = testblock->firstsubblock; tap != NULL; tap = tap->next)
        tap->rect = if_halfcos(tap, testblock);
}

.....
Search the content of the first level nontext block
.....
void firstlevelnontextsearch(p, first)
struct block_rect *first;
{
    struct block_rect *tapb, *tapbb, *tapnnext, *tapl, *tapr;
    float tapl_x, ad, total, total_x;
    int tapl_y, ad_upper, total_y;

    print("get into firstlevelnontextsearch\n");
    for (tapb = first->firstsubblock; tapb != NULL; ) {
        print("index = %d\n", tapb->index);
        if (tapb->firstsubblock == NULL) {
            tapb->firstsubblock = NULL;
            tapb->rect = first;
            print("in firstlevel block in block, x = %d y = %d\n", tapb->rect.x,
                tapb->rect.y);
        }
        if (((tapb->length > 150/SCALE_RATIO) || (tapb->rect.width > (first->width/2)
            || (tapb->rect.height > 150/SCALE_RATIO)))) {
            block_in_block(p, tapb, first);
            block_in_block(p, tapb, first);
        }
    }
}

```

[illegible]

—288—

[illegible]

—289—

【図89】

```

void head_block_into_line(rect_block, head_block, line_block)
{
    struct line_block *line_block;
    struct block_rect *apblock;

    remove_rectblock(rect_block, head_block);
    line_block = new_line_block();
    if (line_block == NULL)
        return;
    line_block->first_block = line_block->current_block = rect_block;
    line_block->previous = rect_block->next = NULL;
    line_block->right_x = rect_block->right_x;
    line_block->upper_y = rect_block->upper_y;
    line_block->lower_y = rect_block->lower_y;
    line_block->numsubblock = 0;
    else {
        line_block->first_block->previous = rect_block;
        rect_block->next = line_block->first_block;
        line_block->first_block->rect_block = rect_block;
        if (rect_block->right_x < line_block->right_x)
            line_block->right_x = rect_block->right_x;
        if (rect_block->upper_y < line_block->upper_y)
            line_block->upper_y = rect_block->upper_y;
        if (rect_block->lower_y > line_block->lower_y)
            line_block->lower_y = rect_block->lower_y;
        line_block->numsubblock++;
    }
}

/*..... test if a gap between two rectblock .....*/
/*..... if gapblock1, block2, headblock, .....*/
struct block_rect *block1, *block2;
int center_x, count, count2;
int len, maxcount, maxcount2;

if (block1->right_x <= block2->right_x-2) return 0;
bound1 = min(block1->upper_y, block2->upper_y);
bound2 = max(block1->lower_y, block2->lower_y);
for (center_x = block1->right_x; center_x < block2->left_x; center_x++)
    for (count = count1; count <= bound2; count++)
        if (center_x <= bound1)
            print("%c", origline_p[count](center_x));
        count++;
    }
    if (count1 == (bound2-bound1+1)) return 1;
}

len = (MAXFONT/SCALE_RATIO + 15*text_len) * MAXFONT/SCALE_RATIO;
maxcount = 0;
maxcount2 = 0;

```

【図90】

```

for (center_x = block1-right_x; center_x < block2->left_x; center_x++)
for (count = 0, i = (center_x < 0 ? 0 : center_x - lower_y); i++) {
    if (original_p[i] < center_block) break;
    else count++;
}
if (count > maxcount) {
    maxcount = count;
    center_x = center_x;
}
if (maxcount > len) {
    len = maxcount;
    center_x = center_x;
}
for (i = (bound_block2)/2; count1 = i; count1 <= maxcount; count1++, i++) {
    return i;
}
else
return 0;

for (count1 = 0, i = upper_y; i <= lower_y; i++)
if (center_x[i] < center_block) count1++;
printf("x = %d upper_y = %d lower_y = %d count1 = %d\n",
center_x, upper_y, lower_y, count1);
if (count1 <= CAPLEN)
return i;
else
return 0;

}

// Stop block into line
// .....
void group_connected_block(stops, column, current_lineblock, headblock, widthfill)
{
    struct lineblock *current_lineblock;
    struct block_rect *headblock;
    struct block_rect *widthfill;
    {
        int x, count, i;
        int firstx, secondx;
        struct block_rect *tag_block;

        // add the first block forward
        headblock_into_line(widthfill(stops), headblock, current_lineblock);
        for (x = stops-1, tag_block = widthfill(stops); x >= 0, x--) {
            if (widthfill(x) < tag_block) {
                widthfill(x) = tag_block;
                continue;
            }
            count = 0;
            for (i = 0; i <= count; i++) {
                if (widthfill(i) < widthfill(x)) {
                    if (i % 4 < column - 1) {
                        widthfill(i) = widthfill(x);
                        count++;
                    } else break;
                }
            }
            if (count < widthhead(count > 0)) {
                current_lineblock->side_gap = RIGHTCAP;
                break;
            }
            tag_block = widthfill(x);
            widthfill(x) = NULL;
            continue;
        }
    }
}

```

—292—

[illegible]

【図92】

```

return 0;
if (tapline->upper.y > max(firstline->lower.y, secondline->lower.y)
    break;
else if ((if_overlap(tapline->left.x, tapline->right.x,
    firstline->left.x, firstline->right.x) ||
    if_overlap(tapline->left.x, tapline->right.x,
    secondline->left.x, secondline->right.x))
    if_neighbor = 1;
    break;
}
}

if (if_neighbor == 1) {
    print("block %d is blocked by %d\n", id, id2);
    firstline->index, firstline->upper.y, firstline->lower.y,
    secondline->index, secondline->upper.y, secondline->lower.y,
    tapline->index, tapline->upper.y, tapline->lower.y, text_len);
    print("neighbor = %d\n", if_neighbor);
}
if (if_neighbor == 1) {
    (cover_range(firstline->upper.y, firstline->lower.y,
    secondline->upper.y, secondline->lower.y) > (if_overlap))
    (firstline->upper.y, firstline->lower.y, secondline->upper.y,
    secondline->lower.y, tapline->upper.y, tapline->lower.y,
    text_len);
    print("firstline = %d secondline = %d tapline = %d\n", firstline->index,
    secondline->index, tapline->index);
    return 1;
}
return 0;
}

// sort line
void sort_line(firstlineblock)
{
    struct lineblock *tap;
    struct lineblock *line;
    int num, i, j, stop;
    if (firstlineblock->firstsubline == NULL) return;
    for (num = 0; tap = firstlineblock->firstsubline; tap = tap->next)
        num++;
    if (num > 0) {
        line = new_linepointer(0, num-1);
        for (i = 0; i < num; i++)
            line[i] = tap;
    }
    /* extra testing */
    sort_linepointer(line, num);
    firstlineblock->firstsubline = line[0];
    firstlineblock->currentsubline = line[num-1];
    if (num > 1) {

```


【图 9-4】

```

for (tapwhite = firstblock->firstwhite, tapblack = NULL, tapwhite = tapwhite->next)
    printf("white index = %d x1 = %d y1 = %d x2 = %d y2 = %d\n",
           tapwhite->x, tapwhite->y, tapwhite->upper_y,
           tapwhite->right_x, tapwhite->lower_y);

for (tapwhite = firstblock->firstwhite, i = 1, tapwhite = NULL,
     i++, tapwhite = tapwhite->next)
{
    if (tapline[i]->first_block == NULL){
        tapline[i]->left_x = (tapwhite->right_x+tapwhite->left_x)/2;
        tapline[i]->right_x = (tapwhite->right_x+tapwhite->left_x)/2;
        tapline[i]->upper_y = tapwhite->upper_y+1;
        tapline[i]->lower_y = tapwhite->lower_y-1;
    }

    if (((tapline[i]->lower_y-tapline[i]->upper_y)< HORIZONTAL_RATIO/4
        for (y = tapline[i]->upper_y-1, j = 1; (j < STANDARD_FONT_HEIGHT) && (y > 0);
            y--, j++){
                for (x = tapline[i]->right_x-x <- tapline[i]->left_x+x; x++)
                    if (original_p[y][x]&0x00ff) break;
                if (original_p[y][x]&0x00ff) break;
            }
        if (tapline[i]->upper_y == y+1) tapline[i]->upper_y=y+2;
        for (y = tapline[i]->lower_y-1, j = 1; (j < STANDARD_FONT_HEIGHT) && (y > 0);
            y--, j++){
                for (x = tapline[i]->right_x-x <- tapline[i]->left_x+x; x++)
                    if (original_p[y][x]&0x00ff) break;
                if (original_p[y][x]&0x00ff) break;
            }
        if (tapline[i]->upper_y <= y) tapline[i]->lower_y = y-2;
    }

    for (x = tapline[i]->left_x, x = tapwhite->left_x+1, x--
         for (y = tapline[i]->lower_y-2, y++
             if (original_p[y][x]&0x00ff)
                 tapline[i]->left_x = x;
                 break;
            )
        for (x = tapline[i]->right_x, x = tapwhite->right_x-1, x++
             for (y = tapline[i]->upper_y+2, y--
                 if (original_p[y][x]&0x00ff)
                     tapline[i]->right_x = x;
                     break;
            )

    tapline[i]->length = tapline[i]->lower_y-tapline[i]->upper_y+1;
    tapline[i]->width = tapline[i]->right_x-tapline[i]->left_x+1;
}

firstlineblock->firstsubline = tapline[1];
firstlineblock->currentsubline = tapline[firstblock->group];
for (tapline[i]->left TEXT;
     tapline[i]->index = 1;
     tapline[i]->next = tapline[i+1];
     for (tap = firstblock->firstlinecenterblock, tap = NULL, tap = tap->next; (
         if (tap->size != (HORIZONTAL_RATIO*HORIZONTAL_RATIO)) break;
         struct block *next;
         int n, y, count;

```

[illegible]

【図96】

```

        tapline=tapline->next){
    for ( i = 0; i <= (level-1)*8; i++)
        fprintf(line_file, " ");
    fprintf(line_file, "%d nontext index = %d ", tapline, tapline->index);
    fprintf(line_file,
"x%d y%d l%d w%d att = %x block index1=%d index2=%d g = %d l = %d r = %d\n",
    tapline->left_x, tapline->upper_y, tapline->length,
    tapline->width, tapline->att, tapline->first_block->index,
    tapline->current_block->index, tapline->group, tapline->l_caption,
    tapline->r_caption);
    print_ln(nextlevel, tapline);
}

.....
..... Open the file for the output of line message .....
.....
void print_lineblock(firstline)
struct lineblock *firstline;
{
    line_file = fopen("line_file.dat", "w");
    print_ln(1, firstline);
    fclose(line_file);
}

```

【図118】

```

        tapsection = tapsection->next){
    if ((tapsection->atts[TITLE|HSOLID|VSOLID|PICTURE])) continue;
    else
        ini_section_block(tapsection, tapsection->firstlineblock, p,
            column);
}

.....
..... Print the line information into a file .....
.....
void print_sec(level, firstsec)
int level;
struct sectionblock *firstsec;
{
    int nextlevel, i;
    struct sectionblock *tapsubsec, *tapsec;
    nextlevel = level+1;
    for (tapsubsec = firstsec->firstsubsec; tapsubsec != NULL;
        tapsubsec = tapsubsec->next){
        for ( i = 0; i <= (level-1)*8; i++)
            printf(" ");
        printf("index = %d ", tapsubsec->index);
        printf("x%d y%d l%d w%d att = %x\n",
            tapsubsec->left_x, tapsubsec->upper_y,
            tapsubsec->length, tapsubsec->width, tapsubsec->att);
    }

    for (tapsec = firstsec->firstnontextsection; tapsec != NULL;
        tapsec=tapsec->next){
        for ( i = 0; i <= (level-1)*8; i++)
            printf(" ");
        printf("nontext index = %d ", tapsec->index);
        printf("x%d y%d l%d w%d att = %x\n",
            tapsec->left_x, tapsec->upper_y, tapsec->length,
            tapsec->width, tapsec->att);
        print_sec(nextlevel, tapsec);
    }
}

.....
..... Open the file for the output of line message .....
.....
void print_secblock(firstsec)
struct sectionblock *firstsec;
{
    print_sec(1, firstsec);
}

```

【図217】

Source Code for Mono-Spaced ("Courier") Segmentation

—298—

```

.....
for ( i = n1; i <= n2; i++)
    v[i].collum_num = 0;
return v;
}
.....
/*----- allocation sectionblockpointer -----*/
struct sect(sectionblock **)max_sect(pointer[n], nh)
{ int n1, nh;
  struct sectionblock **v;
  v = (struct sectionblock **) malloc((nh-nl+1)*sizeof(struct sectionblock**));
  if (!v) {
    printf("error: in the allocation of sectionpointer\n");
    exit(1);
  }
  v = v-nl;
  return v;
}
.....
/*----- sort line block pointer array -----*/
void sort_linepointer(line, num)
{ struct lineblock **line;
  int num;
  int i, j, stop;
  struct lineblock *tmp;
  for ( i = 0; i < num-1; i++){
    for ( j = 0; stop = 0; j < num - i - 1; j++){
      if (line[j]->upper_y > line[j+1]->upper_y){
        tmp = line[j];
        line[j] = line[j+1];
        line[j+1] = tmp;
        stop = 1;
      }
      if (stop == 0) break;
    }
  }
}
.....
/*----- line block sorting -----*/
struct lineblock **sort_v(firstlineblock, numline;
int *numline;
{ struct lineblock *tmp;
  struct lineblock **line_x;
  int i, j, stop;
  *numline = 0;
  for ( tmp = firstlineblock->firstlineblock; tmp != NULL; tmp = tmp->next)
    (*numline)++;
  for ( tmp = firstlineblock->firstlineblock; tmp != NULL; tmp = tmp->next)
    (*numline)--;
  line_x = new_linepointer(0, (*numline)-1);
}
.....
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include "blocksection.h"
.....
Author: Shin-Yuan Wang
Date : 1-8-97
.....
float tcost()
extern (int txt_length;
unsigned char *original_b;
int mny, maxy;
struct segment_gap segment;
int current_seg = 0;
)
.....
/*----- absolute difference of two integers -----*/
int abs(a, b)
{ int s, bi;
  return (abs(b-a));
}
.....
/*----- allocation lineblockpointer -----*/
struct lineblock **new_linepointer(nl, nh)
{ struct lineblock ***;
  v = (struct lineblock **) malloc((nh-nl+1)*sizeof(struct lineblock**));
  if (!v) {
    printf("error: in the allocation of linepointer\n");
    exit(1);
  }
  v = v-nl;
  return v;
}
.....
/*----- allocate segment gap -----*/
struct segment_gap *new_segment(al, nh)
{ int n1, nh;
  struct segment_gap *v;
  int i;
  v = (struct segment_gap *) malloc((nh-nl+1)*sizeof(struct segment_gap));
  if (!v) {
    printf("error: in the allocation of segment_gap\n");
    exit(1);
  }
  v = v-nl;
}

```

【 98】

```

for (i = 0; tap = firstlineblock->firstsubline; tap != NULL; tap = tap->next)
    line_x[i++] = tap;

/* extra testing */
if (firstlineblock->currentsubline != NULL)
    if (firstlineblock->currentsubline->index)
        printf("error in sort.\n");
for (tap = firstlineblock->firstsubline; tap != NULL; tap = tap->next)
    line_x[i++] = tap;
return line_x;
}

/*----- section block sorting -----*/
struct sectionblock *sort_section(first, num)
int num;
{
    struct sectionblock *tmp;
    struct sectionblock *line_x;
    int i, j, stop;

    num = 0;
    for (tap = first->firstsubsection; tap != NULL; tap = tap->next)
        for (num++;

    line_x = new_sectionpointer(0, (num+1));
    for (i = 0; tap = first->firstsubsection; tap != NULL; tap = tap->next)
        line_x[i++] = tap;

    for (i = 0; i < (num+1); i++)
    {
        for (j = 0; stop = 0; j < (num-i+1); j++)
            if ((line_x[i]->upper_y > line_x[j+1]->upper_y) ||
                ((line_x[i]->upper_y == line_x[j+1]->upper_y) &&
                 (line_x[i]->lower_y > line_x[j+1]->lower_y)))
            {
                tmp = line_x[j+1];
                line_x[j+1] = line_x[i];
                line_x[i] = tmp;
                stop = 1;
            }
        if (stop == 0) break;
    }
    return line_x;
}

/*----- Allocate new line block -----*/
struct sectionblock *new_sectionblock(int, int)
int ni, nh;
{
    int i;
    struct sectionblock *v;
    v = (struct sectionblock *) malloc((unsigned)(nh+ni));
    if (!v)
        printf("allocation error: section block %d", ni);
    exit(1);
}

```

—300—

```

if (current->firstlineblock == line)
    current->currentlineblock = line;
else if (current->currentlineblock == NULL)
    current->currentlineblock = NULL;
else
    current->firstlineblock->previous = NULL;
} else if (current->currentlineblock == line) {
    current->current->next = NULL;
} else {
    line->previous->next = line->next;
    line->next->previous = line->previous;
    line->previous = NULL;
    line->next = NULL;
}

..... put linelock into section .....
void pg_section(line, cursection)
struct linelock *line;
cursection: cursection->action();
if (cursection->firstlineblock == NULL) {
    cursection->firstlineblock = line;
    line->previous = NULL;
    line->next = NULL;
} else {
    cursection->currentlineblock->next = line;
    cursection->currentlineblock = line;
    line->next = NULL;
}

..... create new sameline structure .....
struct sam_line *new_sameline(cur)
struct linelock *cur;
{
    struct sameline *v;
    v = (struct sameline *) malloc(sizeof(struct sameline));
    if (!v) printf("allocation error in sameline\n");
    return v;
}

v->current = cur;
v->previous = NULL;
v->next = NULL;
v->connection = v->next->connection = NULL;
return v;
}

..... sort sameline array by x .....
void sort_sameline(first)
struct sameline *first;
{
    ..... combination of line block forward .....
    void fline_combine(rearward, line_train)
    struct linelock *rearward, *line_train;
}

```


【図100】

```

newhead->first_block->previous = line_tran->current_block;
line_tran->current_block->next = newhead->first_block;
newhead->first_block = line_tran->first_block;
newhead->main_block = line_tran->main_block;
if (line_tran->left_x < newhead->left_x)
    newhead->left_x = line_tran->left_x;
if (line_tran->right_x > newhead->right_x)
    newhead->right_x = line_tran->right_x;
if (line_tran->upper_y > newhead->upper_y)
    newhead->upper_y = line_tran->upper_y;
if (line_tran->lower_y < newhead->lower_y)
    newhead->lower_y = line_tran->lower_y;
newhead->width = newhead->left_x - newhead->right_x;
newhead->length = newhead->upper_y - newhead->lower_y;
}

/*..... combination of line block and head .....*/
void bline_combine(newhead, line_tran)
struct lineblock *newhead, *line_tran;
{
    newhead->current_block->next = line_tran->first_block;
    line_tran->first_block->previous = newhead->current_block;
    newhead->current_block = line_tran->current_block;
    if (line_tran->left_x < newhead->left_x)
        newhead->left_x = line_tran->left_x;
    if (line_tran->right_x > newhead->right_x)
        newhead->right_x = line_tran->right_x;
    if (line_tran->upper_y > newhead->upper_y)
        newhead->upper_y = line_tran->upper_y;
    if (line_tran->lower_y < newhead->lower_y)
        newhead->lower_y = line_tran->lower_y;
    newhead->width = newhead->left_x - newhead->right_x;
    newhead->length = newhead->upper_y - newhead->lower_y;
}

/*..... line remove from one head and merge with another head .....*/
void line_remove(line_tran, originalhead, newhead, att)
struct lineblock *line_tran, *originalhead, *newhead, att;
int att;
{
    line_tran->att = originalhead->line_tran;
    bline_combine(newhead, line_tran);
    free((struct lineblock*) line_tran);
}

/*..... test if one line fits inside another line .....*/
int if_inside(firstline, secondline)
{
    if ((float)cover_range(firstline->upper_y, firstline->lower_y,
        secondline->upper_y, secondline->lower_y) > 1.0) &&
        ((float)cover_range(firstline->left_x, firstline->right_x,
            secondline->left_x, secondline->right_x) > 1.0))
        return 1;
    return 0;
}

/*..... cover part .....*/
float cover_part(newhead, a1, a2)
int a1, a2;
{
    int newhead, i;
    for (i = a1; newhead < 0; i <= a2; i++)
        if (mark[i] > 0)
            mark[i] = 1;
    return ((float)upmark/(float)(a2-a1+1));
}

/*..... the number of neighbor on the left side .....*/
void text_left_neighbor(line, leftlevel, leftlength, head, line1st, nodistroy)
struct lineblock *line;
int *leftlevel;
int *leftlength;
struct lineblock *line1st;
struct lineblock *nodistroy;
{
    int a, upper, lower, mark, nextleft;
    struct lineblock *mainline, *mainline, *top;
    upper = line->current->upper_y;
    lower = line->current->lower_y;
    for (a = line->previous; top != NULL; top = top->previous)
        if (top->current->upper_y < upper)
            upper = top->current->upper_y;
        if (top->current->lower_y > lower)
            lower = top->current->lower_y;
    mark = (int *)vector(upper, lower);
    mainline = line->current->left_x;
    leftlevel = 0;
    leftlength = 0;
    for (a = top = line->previous; top != NULL; top = top->previous)
        if ((top->current == NULL) || (top->current->upper_y < upper ||
            top->current->lower_y > lower))
            continue;
        if (line->current->upper_y < top->current->upper_y ||
            line->current->lower_y > top->current->lower_y)
            continue;
        if ((a = cover_part(mainline, top->current->upper_y, top->current->lower_y)) > OVER)
            continue;
        if (top->current->left_x < mainline)
            *leftlevel++ = (float)top->current->left_x;
        if (a != 0.0)
            if (*leftlevel)++
                if (mainline <= NULL) || (a1 <= mainline <= NULL) ||
                    mainline < top;
            mainline = top;
}

```

[101]

```

int upper, lower, start, nostight;
struct sashline *ashline, *ashline, *tap;
float w;

upper = line->current->upper_y;
lower = line->current->lower_y;
for (tap = line->next; tap != NULL; tap = tap->next) {
    if (tap->current == NULL) continue;
    if (tap->current->upper_y < upper) upper = tap->current->upper_y;
    if (tap->current->lower_y < lower) lower = tap->current->lower_y;
}

mark = (int *)vector(upper, lower);
nostight = line->current->right_x;
ashline = ashline = NULL;
rightlength = 0;
for (tap = line->next; tap != NULL; tap = tap->next) {
    if (tap->current == NULL) continue;
    if (line->current->right_x < tap->current->right_x)
        line->transfer(tap->current, head, line->current->next);
    tap->current = NULL;
    line->list[tap->index] = NULL;
    continue;
}
if ((tap->current->left_x < nostight) || (a >= OVERLAP))
    (cover_range(tap->current->upper_y, tap->current->lower_y,
    line->current->upper_y, line->current->lower_y) > 0.0) {
    cover_start(tap->current->upper_y, tap->current->lower_y);
    printf("a = %f", (float)tap->current->length);
    if ((tap->current->right_x < nostight) || (a >= OVERLAP))
        if (tap->current->right_x > nostight)
            nostight = tap->current->right_x;
        else if ((float)tap->current->length >
            (float)tap->current->length,
            if ("rightlevel"))
                if (ashline == NULL) { if (ashline == NULL) {
                    ashline = tap;
                } else if (tap->current->length > ashline->current->length)
                    ashline = tap;
                else
                    if (tap->current->length < ashline->current->length)
                        ashline = tap;
            }
        }
    }
    printf("right level in right_text to %d", rightlevel);
    if (rightlevel == 1)
        if (tap->current->right_x < ashline->current->right_x)
            ashline->current->right_x = tap->current->right_x;
        if (ashline->current->width < ashline->current->length)
            rightlength = (float)ashline->current->length;
        if (ashline->current->length < (float)ashline->current->length)
            (ashline->current->length < ashline->current->length + 2)
            (ashline->current->width < ashline->current->length + 2)
            (rightlevel = 1;
            if (nodistroy) {
                line->transfer(ashline->current, head, ashline->current);
                ashline->current = NULL;
                line->list[ashline->index] = NULL;
                ashline = NULL;
            }
        }
    }
} else if (tap->current->length > ashline->current->length)
    ashline = tap;
    else
        if (tap->current->length < ashline->current->length)
            ashline = tap;
    }
}

if (rightlevel == 1)
    if (nodistroy) {
        line->transfer(ashline->current, head, ashline->current);
        ashline->current = NULL;
        line->list[ashline->index] = NULL;
        ashline = NULL;
    }
} else {
    if (rightlevel == 1)
        if (ashline->current->left_x < ashline->current->left_x) {
            leftlength = ashline->current->length;
            ashline = NULL;
        } else {
            if (leftlength < ashline->current->length)
                ashline = NULL;
        }
    }
}

if (rightlevel == 1)
    line->connection = (ashline == NULL) ? ashline : ashline;

if (rightlevel == 2) { leftlength = (float)leftlevel;
    free(left);
}

/***** the number of tap on the right side *****/
/***** the number of tap on the right side *****/
/***** the number of tap on the right side *****/

void text_right_neighbor(line, rightlevel, rightlength, head, line->list, nodistroy);
struct sashline *line;
int rightlevel;
float rightlength;
struct sashline *head;
struct sashline *nodistroy;

```

【図102】

```

        } else {
            line_transfer(maxline->current, head, minline->current,
                minline->current->next);
            maxline->current = NULL;
            minline->current->next = NULL;
            minline = NULL;
        }
    } else {
        "rightlevel" == 1;
        if (minline->current->next_x < maxline->current->next_x) {
            "rightlength" = minline->current->length;
            maxline = NULL;
        } else {
            "rightlength" = maxline->current->length;
            minline = NULL;
        }
    }
}

if ("rightlevel" == 1)
    line->connection = (maxline==NULL)?minline:maxline;
if ("rightlevel" == 2) "rightlength" /% (float)"rightlevel",
    fmax((int 0) (math:upper));
}

//..... test neighbor of some unknown block
//.....
void test_neighbor(line, leftlevel, rightlevel, leftlen, rightlen, first, line, list, node)
{
    struct maxline *line;
    int leftlevel, rightlevel;
    struct lineblock *first;
    struct lineblock *list;
    struct maxline *node;
    {
        test_left_neighbor(line, leftlevel, leftlen, first, line, list, node);
        test_right_neighbor(line, rightlevel, rightlen, first, line, list, node);
    }
}

//..... make title count left direction
//.....
int right_title(currentline, line, first)
{
    struct maxline *currentline;
    struct lineblock *line;
    struct lineblock *first;
    struct maxline *node;
    {
        struct maxline *currentline;
        int rightmost, rightlevel, leftlevel;
        float rightlength, leftlength;
        int combine;
        combine = 1;
        rightmost = currentline->current->right_x;
        for ( tapline = currentline->connection; tapline != NULL;
            tapline = tapline->next_connection ) {
            if (tapline->current == NULL) continue;
            print("tapline to currentline: %d\n", tapline->current->index,
                currentline->current->index);
            if (tapline->current->next_x == rightmost-tapline->current->length) {
                line;
                test_left_neighbor(tapline, leftlevel, leftlength, first, line, current
                    line);
                print("tapline->next to leftlevel: %d\n",
                    tapline->next_connection->current->index, leftlevel);
            }
            if (leftlevel == 1) {
                if (tapline->next_connection == currentline) {
                    if (cover_range(tapline->current->super_y,
                        tapline->current->lower_y, currentline->current->super_y,
                        currentline->current->lower_y) == 0.5) {

```

【図103】

```

if (tapline->current->right_x > rightmost)
    rightmost = tapline->current->right_x;
line_transfer(tapline->current, first, currentline->current,
    tapline->current->next);
currentline->current->next = first;
tapline->current = NULL;
line(tapline->index) = NULL;
text_right_neighbor(currentline, rightlevel, rightlength,
    first, line, NULL);
if (rightlevel == 1)
    continue;
else
    break;
} else {
    tapline->connection = NULL;
    currentline->current->next = TEXT;
    break;
} else {
    tapline->connection = NULL;
    break;
} else break;
}

//***** pack a set of line block
//*****
void pack_line(line, num)
int num;
{
    int i, j, pack;
    struct lineblock **tapline;
    tapline = new_linepointer(0, (num-1));
    for (i = 0; i < num; i++) {
        if (line[i] == NULL)
            pack = 1;
        else
            tapline[j++] = line[i];
    }
    if (pack == 1) {
        for (i = 0; i < j; i++) line[i] = tapline[i];
        for (i = j; i < num; i++) line[i] = NULL;
        num = j;
    }
    free(struct linepointer **) tapline;
}

//***** index for combining two baseline block
//*****
int index_of_combine(line1, line2)
struct lineblock *line1, *line2;
{
    float l1, m, p, q;

    l1 = cover_range(line1->upper_y, line1->lower_y, line2->upper_y,
        line2->lower_y);
}

```

```

n = cover_range(line1->upper_y, line1->lower_y, line2->upper_y,
    line2->lower_y);
p = cover_range(line1->right_x, line2->right_x, line2->left_x,
    line2->left_x);
q = cover_range(line1->left_x, line2->right_x, line2->left_x,
    line2->right_x);
if ((line1->next != TEXT) && (line2->next != TEXT)) {
    if ((n < 0.5) || (m < 0.5)) return 1;
    else return 0;
}
if ((line1->next == TEXT) && (line2->next != TEXT)) {
    if (m < 0.5) return 0;
    if (m < 1.0/3.0) return 2;
    else return 1;
}
if ((line1->next != TEXT) && (line2->next == TEXT)) {
    if (n < 0.5) return 0;
    if (n < 1.0/3.0) return 2;
    else return 1;
}
if ((line1->next == TEXT) && (line2->next == TEXT)) {
    if ((l1 < 0.5) || (m < 0.5)) return 1;
    else return 0;
}
if (l1 < m) {
    if ((l1 < 1.0/3.0) && (line2->next != TEXT)) return 0;
    else return 1;
}
else {
    if (m < 1.0/3.0) && (line return 0;
    else return 1;
}
}

//***** Combine touched line element inside the baseline
//*****
void inside_test(first, head, line1)
struct lineblock *first,
    struct lineblock *head;
struct lineblock **line1;
{
    int m;
    struct baseline *tap, *tap1;
    struct baseline *tap2;
    printf("before inside test\n");
    for (tap = first; tap != NULL; tap = tap->next) {
        if (tap->current == NULL) printf("null\n");
        else printf("index = %d x = %d y = %d\n", tap->index,
            tap->current->right_x, tap->current->right_y,
            tap->current->length);
    }
    for (tap1 = first; tap1 != NULL; tap1 = tap1->next) {
        if (tap1->current == NULL) continue;
        if (tap1->current->right_x > tap->current->right_x) break;
        if (tap1->index < tap->index) tap->current->index = tap1->index;
        if ((tap->current->next == TEXT) && (tap1->current->next != TEXT)) {
            if ((tap->current->next != TEXT) && (tap1->current->next != TEXT)) {
                tap->current->next = tap1->current->next;
            }
        }
    }
}

```

[illegible]

[図106]

```

segment(current_seg).begin_column = vector(i, columnst);
segment(current_seg).end_column = vector(i, columnst);
for (i = 1; i < columnst; i++) {
    segment(current_seg).begin_column(i) = segment(i-1);
    segment(current_seg).end_column(i) = segment(i-1);
}
for (j = 1; j < segment(current_seg).column_no; j++)
    printf("by = %d end = %d ", segment(current_seg).begin_column(j),
        segment(current_seg).end_column(j));
    printf("\n");
}

for (tapline = nsection(columnst).firstlineblock; tapline != NULL;
    tapline = tapline->next) {
    for (k = 0; k < columnst; k++)
        tapline->first_x = segment(k);
    break;
    if (k != columnst) {
        for (tap = nsection(k).firstlineblock; tap != NULL; ) {
            if ((cover_range(tap->upper_y, tap->lower_y, tapline->upper_y,
                tapline->lower_y) < 0) ||
                (cover_range(tapline->upper_y, tapline->lower_y, tap->upper_y,
                    tap->lower_y) > OVERLAP)) {
                tapline = tap->next;
                continue;
            }
            take_section(tap, nsection(k));
            if (tap->right_x < tapline->right_x - text)
                if (!line_combine(tapline, tap))
                    else
                        tap = tapline;
                        continue;
                    tap = tap->next;
                }
            }
        }
    }

    for (k = 0; k < columnst; k++)
        if ((tapline->right_x < segment(k)) ||
            (tapline->right_x > segment(k)))
            break;
    if (k != columnst) {
        for (i = nsection(k).firstlineblock; tap != NULL; ) {
            if ((cover_range(tap->upper_y, tap->lower_y, tapline->upper_y,
                tapline->lower_y) < OVERLAP) ||
                (cover_range(tapline->upper_y, tapline->lower_y, tap->upper_y,
                    tap->lower_y) > OVERLAP)) {
                tapline = tap->next;
                continue;
            }
            if (tap->right_x < tapline->right_x - text)
                if (!line_combine(tapline, tap))
                    else
                        put_section(tap, nsection(columnst));
                        continue;
                        tap = tap->next;
                    }
                }
            }
        }
    }

    for (i = 0; i < columnst; i++)
        printf("section = %d\n", i);
}

```

[illegible]

【図108】

```

/***** remove current section from some head section *****/
/***** remove current section from some head section *****/
void rm_section(head, section)
{
    struct sectionblock *head, *section;

    if (head->currentsection == section) {
        if (section->previous != NULL)
            section->previous->next = NULL;
        else
            head->firstsection = section->previous;
        head->currentsection = section->previous;
        if (head->firstsection == section)
            head->firstsection->previous = NULL;
        else {
            section->previous->previous = section->previous;
            section->previous->next = section->next;
        }
    }

    /***** remove section from some head section *****/
    /***** remove section from some head section *****/
    void rm_sectionblock(head, section)
    {
        if (head->currentsection == section) {
            if (section->previous != NULL)
                section->previous->next = NULL;
            else
                head->firstsection = section->previous;
            head->currentsection = section->previous;
            if (head->firstsection == section)
                head->firstsection->previous = NULL;
            else {
                section->previous->previous = section->previous;
                section->previous->next = section->next;
            }
        }
    }

    /***** combine two section *****/
    /***** combine two section *****/
    void combine_section(first, second, head)
    {
        struct sectionblock *first, *second, *head;

        print("combine id = %d %d", second->index, first->index);
        first->firstlineblock->previous = second->currentlineblock;
        second->currentlineblock->next = first->firstlineblock;
        if (first->firstlineblock->next != first->firstlineblock)
            first->firstlineblock->next->previous = first->firstlineblock;
        if (first->second->lower_y > first->lower_y)
            first->second->lower_y = first->lower_y;
        if (first->second->left_x < first->left_x)
            first->second->left_x = first->left_x;
        if (first->second->right_x > first->right_x)
            first->second->right_x = first->right_x;
        if (first->second->width > first->width)
            first->second->width = first->width;
        if (first->second->length > first->length)
            first->second->length = first->length;
        first->firstlineblock->next = second->firstlineblock;
    }
}

```


【図110】

```

        (tap->right_x >= min(topsec->left_x, bottomsec->left_x)))
        return 0;
    if (if_overlap(tap->upper_y, tap->lower_y,
        min(topsec->upper_y, bottomsec->upper_y),
        max(topsec->lower_y, bottomsec->lower_y))) {
        if (if_overlap(tap->left_x, tap->right_x,
            min(topsec->left_x, bottomsec->left_x),
            max(topsec->right_x, bottomsec->right_x)))
            return 1;
    }
    return 0;
}

/*
 * ..... Test if some section fully inside the other section .....
 * ..... Test if two sections are neighbors (belong to same column) .....
 */
int full_inside(first, second)
struct sectionblock *first, *second;
{
    if ((first->left_x >= second->left_x) && (first->right_x <= second->right_x) &&
        (first->upper_y >= second->upper_y) && (first->lower_y <= second->lower_y))
        return 1;
    else
        return 0;
}

/*
 * ..... Test if two sections are neighbors (belong to same column) .....
 */
int same_section(first, second, sec, num)
struct sectionblock *first, *second;
int first, sec, num;
{
    int x1, y1, x2, y2, i, j, count;
    struct sectionblock *tap1, *tap2;
    int result_x1, result_x2, result_y1, result_y2;
    char p;

    printf("in same section index 1 = %d index 2 = %d\n", first->first_index,
        second->first_index);
    if (sort_y(first->left_x < sort_y(second->left_x))
        tap1 = sort_y(first);
        tap2 = sort_y(second);
    ) else {
        tap1 = sort_y(first);
        tap2 = sort_y(second);
    }

    if (if_overlap(tap1->upper_y, tap1->lower_y,
        tap2->upper_y, tap2->lower_y) == 0) return 0;

    if (tap1->first_index != tap2->first_index) {
        if (tap1->first_index < tap2->first_index) {
            return 0;
            tap2->first_indexblock-under_section();
        }
        else if (tap1->first_index > tap2->first_index) {
            return 0;
        }
    }
}

```

```

        in_1segment = tap1->boundary_index;
        in_2column = 0;
        in_1segment = tap2->boundary_index;
        in_2column = 0;

        for (j = 1; j = in_1segment; j <= segment[i].column_num; j++) {
            print(i) = %d, x1 = %d, x2 = %d, column_x1 = %d, column_x2 = %d, in_1, j;
            tap1->left_x, tap1->right_x, segment[i].begin_column[j], segment[i].end_column[j];
            if (cover_range(tap1->left_x, tap1->right_x,
                segment[i].begin_column[j], segment[i].end_column[j]) <= 1.0) {
                in_2column = j;
                break;
            }
        }

        print("seg1 = %d col1 = %d seg2 = %d col2 = %d\n", in_1segment, in_2column,
            in_1segment, in_2column);

        for (j = 1; j = in_1segment; j <= segment[i].column_num; j++) {
            if (cover_range(tap1->left_x, tap1->right_x,
                segment[i].begin_column[j], segment[i].end_column[j]) <= 1.0) {
                in_2column = j;
                break;
            }
        }

        print("seg1 = %d col1 = %d seg2 = %d col2 = %d\n", in_1segment, in_2column,
            in_1segment, in_2column);

        if ((in_1segment > 0) && (in_2segment > 0)) {
            if ((in_1segment != in_2segment))
                printf("different segment\n");
            if ((in_2column > 0) && (in_2column != 0)) {
                if (cover_range(tap1->left_x, tap1->right_x,
                    segment[in_2segment].begin_column[in_2column],
                    segment[in_2segment].end_column[in_2column]) <= 1.0) {
                    (cover_range(tap1->left_x, tap1->right_x,
                        segment[in_2segment].begin_column[in_2column],
                        segment[in_2segment].end_column[in_2column]) <= 1.0)) {
                        return 0;
                    }
                } else {
                    return 0;
                }
            } else {
                if ((tap1->left_x < tap1->right_x) && (tap2->left_x < tap2->right_x)) {
                    if (tap1->width > (int) (tap2->width)) {
                        if (tap1->width > (int) (tap2->width)) {
                            return 0;
                        }
                    }
                }
                printf("same segment\n");
            }
        } else {
            return 0;
        }

        printf("same segment, same column\n");
        x1 = tap1->right_x;
        x2 = tap1->left_x;
        if (tap1->upper_y > tap2->upper_y) {

```

【 1 1 1 】

```

        struct sectionblock *tapsec;
        int y1, x1, y2, x2;
        float xcover, ycover, xcover2;

        printf("get into text line: first = %d second = %d\n",
               first->index, second->index);

        if (first->upper_y > second->upper_y) {
            tapsec = first;
            first = second;
            second = tapsec;
        }
        x1 = min(first->left_x, second->left_x);
        x2 = max(first->right_x, second->right_x);
        y1 = min(first->upper_y, second->upper_y);
        y2 = max(first->lower_y, second->lower_y);

        for (tapsec = headsec->first->next; tapsec != NULL; tapsec = tapsec->next) {
            printf("td x1 = %d y1 = %d ", tapsec->index, tapsec->left_x, tapsec->upper_y);
            if ((tapsec->first) || (tapsec->second)) continue;
            xcover = cover_range(tapsec->left_x, tapsec->right_x, x1, x2);
            ycover = cover_range(tapsec->lower_y, tapsec->upper_y, y1, y2);
            if (xcover == 0.0) continue;
            xcover2 = cover_range(tapsec->left_x, tapsec->right_x, x1 - (int)text,
                                  x2 - (int)text);
            if ((tapsec->upper_y < first->lower_y) ||
                (tapsec->lower_y > first->upper_y)) return 1;
            if ((tapsec->upper_y > first->lower_y) ||
                (tapsec->lower_y < first->upper_y)) {
                xcover2 = 0.0;
                if (tapsec->right_x > x2) {
                    x1 = tapsec->left_x;
                    x2 = tapsec->right_x;
                }
                continue;
            }
            else return 1;
        }
        printf("xcover2 = %f upper = %d\n", xcover2, first->upper_y - (int)text);
        if ((tapsec->upper_y > first->upper_y) ||
            (tapsec->lower_y < first->lower_y)) {
            x1 = tapsec->left_x;
            y1 = tapsec->upper_y;
            if (tapsec->left_x < x1) {
                x1 = tapsec->left_x;
            }
            if (tapsec->right_x > x2) {
                x2 = tapsec->right_x;
            }
            continue;
        }
        else return 1;
    }
    if (x1 < min(first->left_x, second->left_x))
        second->left_x = x1;
    if (x2 > max(first->right_x, second->right_x))
        second->right_x = x2;
}

```

【図112】

```

    align |= RIGHTTOPLONGER;
    else if (sort_y_section(i) < sort_y_section(j) <= right_x)
        align |= LEFTTOPLONGER;
    if ((float)abs(sort_y_section(i) - right_x)
        <= aligngap)
        align |= RIGHTALIGN;
    if ((align & LEFTALIGN) ||
        (align & RIGHTALIGN) || (align & LEFTTOPLONGER) ||
        (align & RIGHTTOPLONGER) || (align & LEFTALIGN) ||
        (align & RIGHTALIGN))
        if (combine == 2)
            print("i = %d j = %d align = %d\n", i, j, align);
        else if ((cover_range(sort_y_section(i) ->upper_y,
            sort_y_section(j) ->lower_y) > 0.5) ||
            (cover_range(sort_y_section(j) ->upper_y,
            sort_y_section(i) ->lower_y) > 0.5))
            sort_y_section(i) ->lower_y = sort_y_section(j) ->lower_y;
            sort_y_section(i) ->upper_y = sort_y_section(j) ->upper_y;
            print("i = %d j = %d un = %d\n", j, i);
            if (same_section(sort_y_section, j, i, num))
                if_combine = 1;
        }
        print("after same_section\n");
    }
    if ((if_combine == 1) || (if_combine == 2)) {
        if ((cover_range(sort_y_section(i) ->lower_y,
            sort_y_section(j) ->lower_y) > 0.5) ||
            (cover_range(sort_y_section(j) ->upper_y,
            sort_y_section(i) ->upper_y) > 0.5))
            continue;
        if (if_combine == 2) {
            if (boundary_in(first, sort_y_section(j), sort_y_section(i), align))
                continue;
            if (sort_y_section(i) ->lower_y < sort_y_section(j) ->upper_y)
                if ((float)abs(sort_y_section(j) ->lower_y,
                    sort_y_section(i) ->upper_y) > 0.5)
                    continue;
            sort_y_section(i) ->sect |= sort_y_section(j) ->sect;
            combine_section(sort_y_section(i), sort_y_section(j), (first));
            sort_y_section(j) = NULL;
        }
    }
    if ((align & RIGHTALIGN) || (align & LEFTALIGN)) {
        switch (align & RIGHTALIGN) {
            case LEFTALIGN:
                if_combine = 1;
                break;
            case LEFTTOPLONGER:
                if_combine = 0;
                break;
            case LEFTBOTTOMLONGER:
                if_combine = 1;
                break;
            case RIGHTALIGN:
                if_combine = 0;
                break;
        }
    }
}

/*
 * min(sort_y_section(i) ->first_lineblock ->length,
 * sort_y_section(j) ->first_lineblock ->length)
 */
if ((if_overlap(sort_y_section(i) ->left_x, sort_y_section(j) ->right_x,
    sort_y_section(j) ->left_x, sort_y_section(i) ->right_x)) ||
    align == 0)
    if (sort_y_section(i) ->left_x < sort_y_section(j) ->left_x)
        align |= LEFTTOPLONGER;
    else if (sort_y_section(i) ->left_x > sort_y_section(j) ->left_x)
        align |= LEFTBOTTOMLONGER;
    if ((float)abs(sort_y_section(i) ->left_x,
        sort_y_section(j) ->left_x) <= aligngap)
        align |= LEFTALIGN;
    if (sort_y_section(i) ->right_x > sort_y_section(j) ->right_x)
        align |= RIGHTALIGN;
    if (sort_y_section(i) ->right_x < sort_y_section(j) ->right_x)
        align |= RIGHTTOPLONGER;
    if ((float)abs(sort_y_section(i) ->right_x,
        sort_y_section(j) ->right_x) <= aligngap)
        align |= RIGHTALIGN;
    if ((align & LEFTALIGN) || (align & RIGHTALIGN) || (align & LEFTTOPLONGER) ||
        (align & RIGHTTOPLONGER) || (align & LEFTALIGN) || (align & RIGHTALIGN))
        if (combine == 2)
            print("i = %d j = %d align = %d\n", i, j, align);
        else if ((cover_range(sort_y_section(i) ->upper_y,
            sort_y_section(j) ->lower_y) > 0.5) ||
            (cover_range(sort_y_section(j) ->upper_y,
            sort_y_section(i) ->lower_y) > 0.5))
            sort_y_section(i) ->lower_y = sort_y_section(j) ->lower_y;
            sort_y_section(i) ->upper_y = sort_y_section(j) ->upper_y;
            print("i = %d j = %d un = %d\n", j, i);
            if (same_section(sort_y_section, j, i, num))
                if_combine = 1;
        }
        print("after same_section\n");
    }
    if ((if_combine == 1) || (if_combine == 2)) {
        if ((cover_range(sort_y_section(i) ->lower_y,
            sort_y_section(j) ->lower_y) > 0.5) ||
            (cover_range(sort_y_section(j) ->upper_y,
            sort_y_section(i) ->upper_y) > 0.5))
            continue;
        if (if_combine == 2) {
            if (boundary_in(first, sort_y_section(j), sort_y_section(i), align))
                continue;
            if (sort_y_section(i) ->lower_y < sort_y_section(j) ->upper_y)
                if ((float)abs(sort_y_section(j) ->lower_y,
                    sort_y_section(i) ->upper_y) > 0.5)
                    continue;
            sort_y_section(i) ->sect |= sort_y_section(j) ->sect;
            combine_section(sort_y_section(i), sort_y_section(j), (first));
            sort_y_section(j) = NULL;
        }
    }
    if ((align & RIGHTALIGN) || (align & LEFTALIGN)) {
        switch (align & RIGHTALIGN) {
            case LEFTALIGN:
                if_combine = 1;
                break;
            case LEFTTOPLONGER:
                if_combine = 0;
                break;
            case LEFTBOTTOMLONGER:
                if_combine = 1;
                break;
            case RIGHTALIGN:
                if_combine = 0;
                break;
        }
    }
}

```

—314—

[illegible]

【図114】

```

/***** test if located within the picture *****/
/***** test if text inside another text block or postscript block *****/
int is_picture_in_picture(tap, tapt)
{
    struct lineblock *tapl;
    struct sectionblock *taptl;
    int x, y, count;
    unsigned char p;

    p = TEXT_PICTURE;
    for (y = tap->upper_y; y <= tap->lower_y; y++) {
        count = 0;
        for (x = tap->left_x; x <= tap->right_x; x++)
            if (is_picture_in_picture(tap, tapt, x, y)) count++;
        /* printf("count 1 = %d\n", count); */
        if (count == ((int)(HORIZONTAL/SCALE_RATIO)-2)) continue;
        count = 0;
        for (x = tap->left_x; x <= tap->right_x; x++)
            if (is_picture_in_picture(tap, tapt, x, y)) count++;
        /* printf("count 2 = %d\n", count); */
        if (count > ((int)(HORIZONTAL/SCALE_RATIO)-2)) return 1;
    }

    for (y = tap->left_y; y <= tap->right_y; y++) {
        count = 0;
        for (x = tap->upper_x; x <= tap->lower_x; x++)
            if (is_picture_in_picture(tap, tapt, x, y)) count++;
        /* printf("count 3 = %d\n", count); */
        if (count > ((int)(HORIZONTAL/SCALE_RATIO)-2)) return 1;
    }

    return 0;
}

/***** test if text line has some neighbor in some section *****/
int is_line_neighbor(tape, tapl)
{
    struct sectionblock *taps;
    struct lineblock *tapl;
    struct lineblock *tapl_line;

    printf("tapl = %d x1 = %d y1 = %d x2 = %d y2 = %d\n", tapl->index,
        tapl->left_x, tapl->upper_y, tapl->right_x, tapl->lower_y);
    for (taps = first_lineblock->first_lineblock; taps != NULL;
        taps = taps->next) {
        if (tapl_line == taps) continue;
        if (tapl_line->left_x <= tapl->right_x && tapl_line->right_x >= tapl->left_x &&
            tapl_line->upper_y <= tapl->lower_y && tapl_line->lower_y >= tapl->upper_y) {
            if (is_picture_in_picture(tape, tapl, tapl_line->left_x, tapl_line->upper_y))
                if (is_picture_in_picture(tape, tapl, tapl_line->right_x, tapl_line->lower_y))
                    if (is_picture_in_picture(tape, tapl, tapl_line->left_x, tapl_line->lower_y))
                        if (is_picture_in_picture(tape, tapl, tapl_line->right_x, tapl_line->upper_y))
                            return 1;
        }
    }

    return 0;
}

```


[illegible]

【図117】

```

for (j = 1; j <= segment[i].column; j++)
    printf("bg = %d and = %d", segment[i].begin_column[j],
           segment[i].end_column[j]);
    printf("\n");
}

/* section_separate(firstsection), */
printf("before section combine\n");
/* section_combine(firstsection, boundary);
/* section_combine(firstsection, boundary);
inside_text(firstsection);
inside_text(firstsection);
}
free((struct segment *) segment);
free((struct lineblock **) lineblock);
free((struct lineblock **) boundary);
/* section_combine(firstsection, boundary);
/* line_section_combine(firstsection); */

/* Table test */
void move_line_to_section(firstsection, firstline)
struct sectionblock *firstsection;
struct lineblock *firstline;
{
    struct lineblock *tapline;
    struct sectionblock *currentsection;
    int i;

    printf("get into move\n");
    for (i = 0; tapline = firstline->firstnextline; tapline != NULL; i++)
        currentsection = new_section(firstsection->currentlineblock,
                                       currentsection->firstlineblock,
                                       currentsection->left_x,
                                       currentsection->right_x,
                                       currentsection->upper_y,
                                       currentsection->lower_y,
                                       currentsection->width,
                                       currentsection->length,
                                       currentsection->att,
                                       currentsection->next,
                                       currentsection->previous,
                                       currentsection->left_x,
                                       currentsection->right_x,
                                       currentsection->upper_y,
                                       currentsection->lower_y);
    printf("get out of move\n");
}

/* Test if the text stick with the horizontal line */
void in_section_block(firstsection, firstline, p, column)
struct sectionblock *firstsection;
struct lineblock *firstline;
int column;
{
    struct sectionblock *tapsection;
    int n;
    struct lineblock *tapline;

    original_p = p;
    firstsection->firstlineblock = firstsection->currentlineblock = firstline;
    printf("firstline index = %d text = %d nstart = %d group = %d\n",
           firstline->index, firstline->firsttabline, firstline->firstnextline,
           firstline->group);
    if (firstline->firstnextline != NULL) {
        if (firstline->firstnextline->index < column)
            move_line_to_section(firstsection, firstline);
        else
            section_block(firstsection, firstline, column);
        firstline->firstnextline = NULL;
        firstline->firstnextnextline = NULL;
    }

    for (tapsection = firstsection->firstnextsection; tapsection != NULL;
        tapsection = tapsection->firstnextsection)

```

【图 1 1 9】

[illegible]

—320—

[illegible]

—321—

[illegible]

[illegible]

—324—

[illegible]

【図125】

```

.....
..... File: block_util.c .....
..... Author: Shin-Yuan Wang .....
..... Date : 1-22-91 .....
..... Copyright 1991 Canon Information Systems .....
.....
..... test if covered range overlap .....
.....

int if_overlap(a1, a2, b1, b2)
int a1, a2, b1, b2;
{
    if ((a2 < b1) || (b2 < a1)) return 0;
    return 1;
}

..... minimum of two integers .....
.....

int min(a, b)
int a, b;
{
    if (a <= b) return a;
    else return b;
}

..... ratio of overlappoint .....
.....

float cover_range(a1, a2, b1, b2)
int a1, a2, b1, b2;
{
    float range = 0.0;
    int x1, x2;

    if ((a2 < b1) || (b2 < a1)) return range;
    else {
        x1 = max(a1, b1);
        x2 = min(a2, b2);
    }

    range = (float)(x2-x1+1)/(float)(a2-a1+1);
    return range;
}

```

【図131】

```

.....
..... Filename: blockline.h .....
..... Author: Shin-Yuan Wang .....
..... Date : 1-15-92 .....
..... Copyright 1992 Canon Information Systems .....
.....
#define EMPTY 0
#define OVERLAP 1
#define NEXT 2
#define SURROUNDIN 3
#define SURROUNDOUT 4

#define TH 1.7
#define WIDTHSTD 1.1
#define GAPLENGTH 0

#define DOT_BOTTOM 0.9
#define DOT_WIDTH 1
#define LINEDOVERLAP 0.5

#define LEFTGAP 0x10
#define RIGHTGAP 0x01
#define NOISE_NO 3

typedef int fillcondition;

extern void as_draw_box();
extern void convert_to_headblock();
extern void transfer_block();
extern void remove_textblock();
extern int max();
extern int min();
extern float cover_range();
extern struct blockline *new_linepointer();
extern void sort_linepointer();

```

【图 127】

```

.....Filename: block_main.h.....
.....Author: Shin-Yuan Wang.....
.....Date: 1-15-92.....
.....Copyright 1992 Canon Information Systems.....
.....

#define TESTTIME 20
#define TEXT_BLOCK 0x11,
#define LINEPICTURE_BLOCK 0x21
#define NPICTURE_BLOCK 0x31
#define TABLE_BLOCK 0x41
#define LIME_BLOCK 0x51
#define FRAME_BLOCK 0x61
#define UNKNOWN_BLOCK 0x71

#define TEXT_COLOR 0
#define TABLETEXT_COLOR 1
#define LINE_COLOR 2
#define PICTURE_COLOR 3
#define NPICTURE_COLOR 4
#define UNKNOWN_COLOR 5
#define TABLE_COLOR 6
#define FRAME_COLOR 7
#define LINEGAP_COLOR ?

enum {SP_ARROW = 0x01,
      SP_ARCADE = 0x02,
      SP_GOM = 0x04,
      SEP_BLOCK = 0x08};

struct block_image
{
    int width;
    int height;
    unsigned char** pixel;
};

int xoff[TESTTIME][2] = {(0, 1), (1, 2), (0, 1), (1, 2), (0, 1),
                          (1, 2), (0, 0), (0, 0), (1, 1), (1, 1), (1, 1),
                          (2, 2), (2, 2), (1, 0), (2, 3), (1, 0),
                          (3, 1), (0, 1), (0, 1), (1, 1), (1, 3), (1, 3)};

int yoff[TESTTIME][2] = {(0, 0), (0, 0), (1, 1), (1, 1), (2, 2),
                          (2, 2), (0, 1), (1, 2), (0, 1), (1, 2),
                          (0, 1), (0, 2), (0, 1), (0, 1), (1, 2),
                          (1, 3), (1, 2), (0, 1), (1, 2), (0, 1), (1, 1)};

#define TOTALCOLOR 7

char *color_label[]={
    "Text",
    "Text in the table",
    "Picture",
    "Line Drawing Graphics",
    "Solid Line (Vertical or Horizontal)",
    "Table",
    "Frame"};

int color_code[]={TEXT_COLOR, TABLETEXT_COLOR, NPICTURE_COLOR, PICTURE_COLOR,
                  LINE_COLOR, TABLE_COLOR, FRAME_COLOR};

```

【例 132】

```

/* ..... */
/* ..... Filename: blocksection.h ..... */
/* ..... Author : Shin-Twan Wang ..... */
/* ..... Date : 1-28-92 ..... */
/* ..... Copyright 1992 Canon Information Systems ..... */
/* ..... */

#define GAP 16
#define OVERLAP 0.9
#define HOVERLAP 0.5
#define LEVELOVERLAP 0.0
#define LEFTSIDE 0x10
#define LEFTALIGN 0x90
#define LEFTTROPLONGER 0x40
#define LEFTBOTTLONGER 0x20
#define RIGHTSIDE 0x0f
#define RIGHTALIGN 0x08
#define RIGHTTROPLONGER 0xd4
#define RIGHTBOTTLONGER 0xc2
#define TEXTPIXEL 0x0e

#define MOISE_COUNT 5

extern int line_inside();
extern int max();
extern int min();
extern int if_overlap();
extern void xs_draw_box();
extern float cover_range();
void exit();

struct segment_gap
{
    int column_bot;
    int *begin_column;
    int *end_column;
};
```

—327—

[illegible]

—328—

```

(1, SW, SOUTH, SE, EAST, NE, NORTH, NW), /* for SOUTH_EAST */
(6, SE, EAST, NE, NORTH, NW, WEST, S), /* for EAST */
(7, SE, EAST, NE, NORTH, NW, WEST, SW), /* for NORTH_EAST */
(16, NE, NORTH, NW, WEST, SW, SOUTH, S), /* for NORTH */

MAP_jar[8][0] = {
    (SKIP, SKIP, SKIP, SKIP, SKIP, SKIP, SKIP, SKIP),
    (SKIP, LEFT, LEFT, LEFT, LEFT, RIGHT, RIGHT, RIGHT), /* NORTH_WEST */
    (SKIP, LEFT, LEFT, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT), /* WEST */
    (SKIP, SKIP, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT, AL), /* SOUTH_WEST */
    (SKIP, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT, AL), /* SOUTH */
    (SKIP, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT, AL), /* SOUTH_EAST */
    (SKIP, RIGHT, RIGHT, LEFT, LEFT, LEFT, LEFT, LEFT), /* NORTH */
    (SKIP, SKIP, LEFT, LEFT, LEFT, LEFT, LEFT, AL), /* NORTH_EAST */
    (SKIP, LEFT, LEFT, LEFT, LEFT, LEFT, LEFT, AL), /* NORTH */
};

#include "blockrect.h"
Author: Shin-Yuan Wang
Date: 1-15-92
Copyright: 1992 by Information Systems
Recording

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 1000000
#define ONE 1
#define ZERO 0
#define LEFT -ONE
#define RIGHT ONE
#define LEFT_ZERO ZERO
#define RIGHT_ZERO ZERO
#define SOLIDFILED 9.0
#define SOLIDPERCENTITY 0.9
#define FINDERAREA 0.017453406
#define ROT 90
#define WIND_SCALE 2.0
#define WIND_RATIO 1
#define GAP 1
#define COVER 1
#define COVERED 1
#define CATCHAL 2
#define CENTER 0
#define M 1
#define WEST 1
#define SOUTH 4
#define SE 5
#define EAST 6
#define NE 7
#define NORTH 8

#define SKIP 0
#define LEFT 1
#define RIGHT 2
#define SE 3

extern int f_lowerlap();
extern int f_catchal();
extern float cover_range();

struct loc_point
{
    int x;
    int y;
    around_center[9]=({0,0},{-1,-1},{-1,0},{-1,1},
        {-1,1},{0,1},{0,1},
        {1,0},{1,-1},{0,-1});
}

int search_map[9][9]={{{0,0,0,0,0,0,0,0,0},
    {{NE,NORTH,NW,WEST,SW,SOUTH,SE}}, /* for NORTH_WEST */
    {{E,EAST,NE,NORTH,NW,WEST,S}}, /* for NORTH */
    {{NW,WEST,SW,SOUTH,SE,EAST,NE}}, /* for SOUTH_WEST */
    {{S,SOUTH,SE,EAST,NE,NORTH,S}}, /* for SOUTH */
    {{SW,SOUTH,SE,EAST,NE,NORTH,S}}, /* for SOUTH */
    {{SW,SOUTH,SE,EAST,NE,NORTH,S}}, /* for SOUTH */
    {{SW,SOUTH,SE,EAST,NE,NORTH,S}}, /* for SOUTH */
    {{SW,SOUTH,SE,EAST,NE,NORTH,S}}, /* for SOUTH */
}};
```

```
.....  
Filemans: Table.h .....  
Author: Shin-Ywan Wang .....  
Date: 8-15-93 .....  
Copyright 1992 Canon Information Systems .....  
.....  
  
#include "blockgeneral.h"  
  
#define NONE 0  
#define OFFSET 3  
#define OFFSETS 6  
#define NEW 1  
#define OLD 2  
#define NONSTOP 0  
#define STOP 1  
#define TABLE_TESTTIME 2  
  
struct white_for_table  
{  
    int status;  
    struct whiteblock *white;  
    struct white_for_table *previous, *next;  
};
```

```
.....  
..... Filename: images.h .....  
..... Author: Chin-Yuan Wang .....  
..... Date: 1-15-93 .....  
..... Copyright 1992 Canon Information Systems .....  
.....  
  
#include<stdio.h>  
#include<stdlib.h>  
  
struct Images  
{  
    int sizeX;  
    int sizeY;  
    int dpi;  
    unsigned char **pixel;  
};
```

```

*
* Filename: autoinput.h
* Header file for autoinput.c
*
.....
*/

/* Function Definitions for mymalloc.c */
#ifdef CPP
/* Definition for C++ files */
extern "C" {
    void auto_initialize();
    int auto_input_int();
    void auto_terminate();
}
#else
/* Definition for C files */
void auto_initialize();
int auto_input_int();
void auto_terminate();
#endif

```

```

-----*/
void Zerovpp_terminate(nlalist)
  struct NonZerovppList *nlalist; /* The list */
{
  free_vector(nlalist->list); /* Deallocate memory */
  nlalist->allocated = 0; /* Est so won't be accidently used */
  nlalist->size = 0;
}

-----*/

void Zerovpp_add(nlalist, a, b)
  struct NonZerovppList *nlalist; /* The list */
  int a, b; /* Start and stop of the zerovpp */
{
  if (nlalist->size > nlalist->allocated) {
    printf("NonZerovppList FULL!\n");
    exit(1);
  }

  nlalist->list[(nlalist->size)++] = a;
  nlalist->list[(nlalist->size)++] = b;
}

```

```

/* If your routines are in C++ then CPP must be defined in make file */
#ifdef CPP
extern "C" void xs_set_foreground(int x, int y, int b);
extern "C" void xs_set_background(int x, int y, int b);
extern "C" void xs_init();
extern "C" void xs_flush();
extern "C" void xs_plotxy(int x, int y);
extern "C" void xs_unplotxy(int x, int y);
extern "C" void xs_redisplay();
extern "C" void xs_clear_bitmap(int x, int y, int width, int height, char *buff);
extern "C" void xs_clear_win();
extern "C" void xs_draw_box(int x, int y, int width, int height);
extern "C" void xs_string(int x, int y, char *str);
extern "C" void xs_setcolor(int color);
extern "C" void xs_win_dia(int width, int height);
#endif
#endif

```

[illegible]

【図140】

```

    ocl = *(oc1++);
    nc = *(nc1++);
    for (j=0; j<newr; j++)
    {
        temp1 = *(oc1++);
        temp2 = *(oc2++);
        *(nc1++) = (temp1 & *(oc1++)) | (temp2 & *(oc2++));
        if (b) // Do the right end
            *(nc) = *(oc1) | *(oc2) | *(oc3);
    }
    // Do the bottom line (if a == 1 or 2)
    nc = *nc;
    if (a == 1) // a == 1 it's one row
    {
        ocl = *oc1;
        for (j=0; j<newr; j++)
        {
            temp1 = *(oc1++);
            *(nc1++) = temp1 & *(oc1++);
        }
        if (b) // Do the right end
        {
            *(nc) = *(oc1);
            if (a == 2) // a == 2 it's two rows
            {
                ocl = *(oc1++);
                oc2 = *oc2;
                for (j=0; j<newr; j++)
                {
                    temp1 = *(oc1++);
                    temp2 = *(oc2++);
                    *(nc1++) = (temp1 & *(oc1++)) | (temp2 & *(oc2++));
                }
                if (b) // Do the right end
                {
                    *(nc) = *(oc1) | *(oc2);
                }
            }
        }
    }
}

// Do the bottom line (if a == 1)
nc = *nc;
if (a == 1) // a == 1 it's one row
{
    ocl = *oc1;
    for (j=0; j<newr; j++)
    {
        temp1 = *(oc1++);
        *(nc1++) = temp1 & *(oc1++);
    }
    if (b) // Do the right end
    {
        *(nc) = *(oc1);
    }
}

// 1st AND 3rd OR LOGIC
void tline_and_or_3(cimages oldline, cimages newline)
{
    int older, older; // Size of old image
    int newer, newer; // Size of new image
    unsigned char *oc1, *oc2, *oc3, *nc; // The old & new column pointers
    unsigned char *nc1, *nc2; // The old & new row pointers
    register unsigned char temp1, temp2, temp3;

    older = oldline.size;
    older = oldline.size;
    older = (older-1)/2; // Should be ==
    newer = (newer-1)/2; // Should be ==
    newline.allocate(newer, newer);

    a = older - newer; // How much is partially covered in the last row
    if (a == 1) // It's evenly aligned
        newer--;
    b = older - newer; // How much is partially covered in the last column
    if (b == 1)
        newer--;
    b = 0; // == 0 for optimization of ends below
    or = oldline.pixel;
    nr = newline.pixel;
    for (i=0; i<newr; i++) // Do the middle
    {
        oc1 = *(oc1++);
        oc2 = *(oc2++);
        oc3 = *(oc3++);
        for (j=0; j<newr; j++)
        {
            temp1 = *(oc1++);
            temp2 = *(oc2++);
            temp3 = *(oc3++);
            *(nc1++) = (temp1 & *(oc1++)) | (temp2 & *(oc2++)) | (temp3 & *(oc3++));
        }
    }
}

```


【図141】

```

/* filename, autolinput.h
 * This file contains the code to input from a filename
 */
.....
#include <stdio.h>
#include "autolinput.h"
/* Global variables */
int auto_eof_detected;
FILE *auto_fp;
/* The file pointer */
int save_count;
/* Saving the input to a file? */
char filename[1024];
/* The filename */
.....
void auto_initialize()
{
    int not_exit = 1;
    auto_eof_detected = 0;
    while(not_exit)
    {
        printf("Input filename for auto-input? ");
        scanf("%s", filename);
        auto_fp = fopen(filename, "r");
        if (auto_fp == NULL) /* Read open failed */
        {
            auto_fp = fopen(filename, "w");
            if (auto_fp == NULL) /* Write open failed */
            {
                printf("cannot open file %s\n", filename);
            }
            else
            {
                printf("is opened as a new file\n", filename);
                printf("Enter 1 at the next prompt\n");
                not_exit = 1;
                auto_eof_detected = 1;
                not_exit = 0;
            }
        }
        else
        {
            printf("opening existing file %s\n", filename);
            not_exit = 0;
        }
    }
    printf("Do you wish to save the auto-input into this file? (1 = yes) ? ");
    save_to_file = auto_input_int();
    printf("Warning! Change the 1st '1' to a '0' to turn off appending to the file\n");
    save_count = 0;
}

int auto_input_int()
{
    int i;
    if (auto_eof_detected)
    {
        printf("\nManual Input (-9 = newline, -9999 = exit) ? ");
        scanf("%d", &i);
        if (i == -9999)
        {
            fclose(auto_fp);
            exit(1);
        }
        if (save_to_file == 1)
        {
            if (i == -9) /* Print new lines */
            {
                printf(auto_fp, "\n"); /* Print the new lines */
                fflush(auto_fp);
                save_count = 0;
                return auto_input_int(); /* And input again */
            }
            printf(auto_fp, "%d ", i);
            fflush(auto_fp);
            if (++save_count > 20)
            {
                save_count = 0;
                printf(auto_fp, "\n");
            }
            return i;
        }
        if (feof(auto_fp) || fgetc(auto_fp) == EOF)
        {
            printf("\n----- EOP OF FILE DETECTED -----");
            if (save_to_file == 1)
            {
                fclose(auto_fp);
                auto_fp = fopen(filename, "a");
                if (auto_fp == NULL)
                {
                    printf("..... WARNING: CANNOT APPEND TO FILE. Is '%s', filename", filename);
                    save_to_file = 0;
                }
                printf(auto_fp, "\n\n");
                auto_eof_detected = 1;
                return auto_input_int();
            }
            printf("\n\n");
            return i;
        }
    }
}

```

【図143】

```

/* File name: Charlist.c
 * This file contains Charlist and MonZeroVpList utility routines
 */
.....
/*
 * By Christopher Sherrick
 */
.....
#include "charlist.h"
#include "qmalloc.h"

/*
 * Initialize and allocate a Charlist (set up curve pointers)
 * Call this routine once only!
 */
void Charlist_initialize(struct Charlist *clist, int max, vector_height)
{
    struct Charlist *clist; /* The Charlist */
    int clist_max; /* The max number of characters */
    int vector_height; /* Height of the line (to allocate curves) */
    int i; /* A looping var */

    /* Allocate memory for characters */
    clist->chr = (struct Character *) malloc((unsigned) clist_max *
        sizeof(struct Character));
    if (!clist->chr) {
        printf("Memory Allocation Error in Charlist_initialize()\n");
        exit(1);
    }
    clist->size = 0; /* Number of characters */
    clist->allocated = clist_max; /* Size Allocated */
    clist->first = 0; /* No first character */
    clist->vector_height = vector_height;

    /* Allocate memory for curves and set pointers */
    for (i=0; i<clist_max; i++) {
        clist->chr[i].curve = vector(vector_height);
        clist->chr[i].curve = vector(vector_height);
    }

    /* Zero a previously allocated Charlist */
    void Charlist_zero(struct Charlist *clist)
    {
        struct Charlist *clist; /* Number of characters */
        clist->size = 0; /* No first character */
        clist->first = 0;

    }

    /* Terminate (deallocate the Charlist) */
    void Charlist_terminate(struct Charlist *clist)
    {
        struct Charlist *clist;
        int i;

        for (i=clist->allocated; i>=0; i--) { /* Free curves */
            free_vector(clist->chr[i].curve);
            free_vector(clist->chr[i].curve);
        }
    }
}

/* Place a new character in the charlist */
/* Takes the character beyond the charlist (.chr[site]),
 * calculates site, and inserts it into the list */
void Charlist_insert(struct Charlist *clist, struct Character *newchar)
{
    struct Character *newchar; /* pointer to the new character adding */
    newchar = *(&clist->chr[clist->site]); /* Set newchar */
    /* Calculate the position of the new character */
    Charlist_recalculate_loc(newchar);

    /* Check for full list */
    if (clist->site == clist->allocated) /* If 100 big.. */
        printf("Error in Charlist_place_new_char(): Character list is FULL!\n");
        exit(1);

    /* Place new in the list character */
    Charlist_place_character(clist, newchar);

    /* This routine repositions a character in the charlist. It removes
     * the character, recalculates loc and inserts it in the list accordingly.
     */
    void Charlist_replace_char(struct Charlist *clist, struct Character *ch)
    {
        struct Character *ch;
        /* Recalculate the position of the character */
        Charlist_recalculate_loc(ch);
        return;
    }
    if (clist->size < 3) /* If it's the only item in the list */
        return;

    /* Remove the character from the list */
    if (ch->prev == 0) {
        clist->first = clist->first->next; /* If list on list */
    } else {
        clist->first->prev = 0;
    }
    if (ch->next != 0) {
        ch->prev->next = ch->next;
        ch->next->prev = ch->prev;
    }

    /* Place the character in the list */
    Charlist_place_character(clist, ch);

    /* Insert a character after doing a cut */
}

```

【図1-44】

```

/* This routine will insert the character beyond the charlist (charlist).
 * loc is the location of the character and the new character; however
 * loc is not used to place the character. It is only used to
 * place the new character after charlist. After finished, it is an
 * extremely good idea to resort the heaplist!
 */
void Charlist_Insert_After_Charlist(charlist, char)
struct Charlist *charlist; /* Pointer to the new character adding */
struct Character *newchar; /* Set newchar */
{
    /* Calculate the position of the new character */
    Charlist_Precalculate_Loc(newchar);
    Charlist_Precalculate_Loc(charlist);
    /* Check for full list */
    if (charlist->size == charlist->size_allocated) /* If too big... */
    {
        printf("Error in Charlist_Place_New_Char(): Character list is FULL!\n");
        exit(1);
    }
    newchar->prev = charlist;
    newchar->next = charlist->next;
    charlist->next = newchar;
    if (newchar->next != 0)
        newchar->next->prev = newchar;
}

/* This routine will remove a character from the character list
 * Also, the next and next pointers of charlist are not altered */
void Charlist_Remove_Charlist(charlist, ch)
struct Charlist *charlist;
struct Character *ch;
{
    /* Remove the character from the list */
    if (ch->prev == 0) /* If it is on list */
        charlist->first = charlist->first->next;
    else
        charlist->prev->next = 0;
    newchar->next = ch->next;
    if (ch->next != 0)
        ch->next->prev = ch->prev;
}

/* This routine will recalculate the position of a character */
void Charlist_Precalculate_Loc(char) /* Output is the position */
struct Character *ch; /* The character */
{
    int i, loc;
    /* Calculate the position of the new character */
    loc = 0;
    for (i = charlist->top; i != 0; i = charlist->curves[i])
        loc += charlist->curves[i] * ch->curves[i]; /* ... of the curves */
    loc = loc * 4 / (ch->bot - ch->top); /* Set it */
    ch->loc = loc;
}

```

```

/* This routine will place a character (that is not in the charlist)
 * into the charlist.
 */
void Charlist_Place_Character(charlist, ch)
struct Charlist *charlist; /* The character */
struct Character *ch; /* The character */
{
    int loc;
    struct Character *p; /* Used for searching for location */
    struct Character *q; /* The character before p */
    /* Is the list empty? */
    if (charlist->size == 1)
    {
        ch->next = 0;
        ch->prev = 0;
        charlist->first = ch;
        return;
    }
    /* Possible optimization: Don't start from start of list...
     * start from where at */
    loc = ch->loc; /* Retrieve position */
    /* Is it at the beginning of the list? */
    p = charlist->first; /* Load p with the first item on the list */
    if (loc < p->loc) /* If it's the first on the list */
    {
        p->prev = ch;
        ch->next = charlist->first;
        ch->prev = 0;
        charlist->first = ch;
        return;
    }
    /* Set the list on the list to this */
    while (p->loc <= loc) /* until find record past where to place */
    {
        last_p = p; /* Remember last position */
        p = p->next; /* Get the next character */
        if (p == 0) /* If it's the end of list, insert it there */
            break;
    }
    ch->prev = last_p;
    ch->next = last_p->next;
    if (last_p->next != 0)
        p->prev = ch;
    p->prev = ch;
}

/* Allocate memory for list */
/* Compute memory allocated, etc */
size_t size; /* Allocate memory */
void Zero_Vop_Lists_Suff(void) /* Allocate memory */
{
    struct NonZero_Vop_List *nlist; /* The list */
    int size; /* The number of entries (2x allocated) */
    /* Allocate memory for list */
    size = size * 2; /* Allocate memory */
    nlist = (struct NonZero_Vop_List *) malloc(size); /* Set max allocated */
    nlist->size = 0; /* Set to zero elements */
}

```

【図146】

```

/* Filename: charlist.h
 * Header file for charlist.c
 */

.....

/* Define CharTypes: Different types of characters */
typedef enum {
    CHARACTER_RAISED,
    CHARACTER_LOWER_CASE,
    CHARACTER_UPPER_CASE,
    CHARACTER_FULL_OUTLINED,
    CHARACTER_COINED
} CharType;

typedef enum {
    PASSED_RECOGNITION,
    RECOGNITION,
    UNRECOGNIZED
} CharRecognition;

/* Struct CharList:
 * Contains information about an outlined character in a page
 */
struct CharList {
    CharType type;
    CharRecognition rec;
    int level_outlet;
    int loc;
    int top;
    int bot;
    int left;
    int right;
    int curve1;
    int curve2;
    struct Character *next;
    struct Character *prev;
};

/* Struct CharList:
 * Contains information about the list of characters */
struct CharList {
    struct Character *chr;
    int size;
    int allocated;
    int vector_height;
    struct Character *first;
};

.....
/* Zero Vpp List Stuff
 */
.....
/* Struct NonZeroVppList:
 * Contains information about a list for non-zero vpp's. */
.....
struct NonZeroVppList {
    int* list;
    /* Array: In *start, 2n-1 * stop */
    /* Memory allocated (2,4,6,...) */
    /* How many used (1,4,6,...) */
};

.....
/* Define Routines
 */
.....
/* Define C++ Routines */
extern "C" {
    void CharList_initialize(struct CharList *clist, int clist_max,
        int clist_min);
    void CharList_initialize(struct CharList *clist);
    void CharList_terminate(struct CharList *clist);
    void CharList_place_new(struct CharList *clist);
    void CharList_remove(struct CharList *clist);
    void CharList_insert_after(struct CharList *clist,
        struct Character *chr);
    void CharList_remove(struct CharList *clist);
    void CharList_calculate_location(struct Character *chr);
    void CharList_calculate_location(struct CharList *clist, struct Character *chr);
    void CharList_initialize(struct NonZeroVppList *nvlst, int size, int b);
    void CharList_terminate(struct NonZeroVppList *nvlst, int size, int b);
    void CharList_add(struct NonZeroVppList *nvlst, int size, int b);
};

.....
/* Define F Routines */
void CharList_initialize();
void CharList_terminate();
void CharList_place_new(struct Character *chr);
void CharList_remove(struct Character *chr);
void CharList_insert_after(struct CharList *clist, struct Character *chr);
void CharList_remove(struct CharList *clist);
void CharList_calculate_location(struct Character *chr);
void CharList_calculate_location(struct CharList *clist, struct Character *chr);
void CharList_initialize(struct NonZeroVppList *nvlst, int size, int b);
void CharList_terminate(struct NonZeroVppList *nvlst, int size, int b);
void CharList_add(struct NonZeroVppList *nvlst, int size, int b);
};

```

[illegible]

[図148]

```

// Make the new climage (note also adjusted)
for (int i=0; i<sizeR; i++)
    for (int j=0; j<sizeC; j++)
        pixel[i][j] = value;

// Copy section of C1 into another. image = reference image
void climage::clip(climage& image, int R1, int R2, int C1, int C2)
{
    int r,c,t;

    if (R2 < R1)
        // Swap R1 & R2
        if (C2 < C1)
            // Swap C1 & C2
            if (R1 < 0) R1 = 0;
            if (R2 < 0) R2 = 0;
            if (C1 < 0) C1 = 0;
            if (C2 < 0) C2 = 0;
            if (R1 > image.sizeR) R1 = image.sizeR;
            if (R2 > image.sizeR) R2 = image.sizeR;
            if (C1 > image.sizeC) C1 = image.sizeC;
            if (C2 > image.sizeC) C2 = image.sizeC;
            allocate(R2-R1, C2-C1);
            cpl = image.cpl;
            for (t=0; t<R2-R1; t++)
                for (c=0; c<C2-C1; c++)
                    pixel[t][c] = image.pixel[r+R1][c+C1];
}

// Clipimage: reduce the size of an image by removing the black
// space around it. This adjusts the image size.
void climage::clipimage()
{
    int R1, R2, C1, C2; // The starting and stopping rows of new image
    int i, j; // Temporary image
    struct climage temp; // Temporary image

    R1 = sizeR;
    R2 = 0;
    C1 = 0;
    C2 = 0;

    // Capture dimensions of new image
    for (i=0; i<sizeR; i++)
        if (pixel[i][0] != 0)
            if (i < R1) R1 = i;
            if (i > R2) R2 = i;
            if (i < C1) C1 = i;
            if (i > C2) C2 = i;
            if (i > C3) C3 = i;

    if (R1 > R2) R1 = R2 = 0;
    if (C1 > C2) C1 = C2 = 0;

    temp.allocate(R2-R1+1, C2-C1+1); // New image is this size

```

```

// Copy clipped image
for (j=0; j<sizeC; j++)
    temp.pixel[i-R1][j-C1] = pixel[i][j];

// Copy temp image to current image
for (j=0; j<sizeC; j++)
    pixel[i][j] = temp.pixel[i][j];
}

// Read in part of a TIFF file
void climage::readTIFF(char *filename)
{
    FILE *tagfile; // Input file
    long int tagfile_position; // The our file position (in bytes)
    long int length; // Length of picture
    long int black; // Value of black pixel
    long int bps; // Bytes per strip
    long int raster_offset; // Start of image data
    long int numtags, tag_type, len, value; // Tag values
    long int i;

    if (!tagfile = fopen(filename, "rb+")) -- NULL;
    curr << "Error: in opening " << filename << "\n";
    exit(1);

    tagfile_position = 0;

    // Read in header and make sure it's Intel
    i = getbytes(tagfile, tagfile_position);
    if (i != 0x00000001)
        curr << "... Error: TIFF File not in Intel Format\n";
    exit(1);

    // Read in magic number
    i = getbytes(tagfile, tagfile_position);
    if (i != 0x002A)
        curr << "Bad Magic Number for TIFF file!\n";
    exit(1);

    // Find first file directory
    j = getbytes(tagfile, tagfile_position);
    while (tagfile_position < j)
    {
        i = getbytes(tagfile, tagfile_position);
        numtags = getbytes(tagfile, tagfile_position);
        while (--numtags > 0)
        {
            tag = getbytes(tagfile, tagfile_position);
            type = getbytes(tagfile, tagfile_position);
            len = getbytes(tagfile, tagfile_position);
            value = getbytes(tagfile, tagfile_position);
            if ((len & 1) && (tag & 32768) == 0)
                curr << "Error: Tag Length is 1\n";
        }
    }
}

```

【図149】

```

    } // while
    while(tagfile_position < raster_offset)
    {
        i = get2byte(tagfile_tagfile_position);
        //----- READ IN FILE -----
        allocate((int) length, (int) width);
        int v;
        for (i = 0; i < length; i++)
        {
            for (j = 0; j < width; j++)
            {
                if ((j%8) == 0)
                {
                    v = get2byte(tagfile);
                    tagfile_position++;
                    if (v == 255)
                    {
                        next << "An Error: Premature End of File found in tiff file!\n";
                        exit(1);
                    }
                    if ((v & 128) == black+128)
                    {
                        pixel[i][j] = 0;
                    }
                    else
                    {
                        pixel[i][j] = 1;
                    }
                    v = v << 1;
                }
            }
        }
        fclose(tagfile);
    }
    //----- READ IN FILE -----
    // Read in a byte. File position is updated
    // eoferror = 1 to error on EOF, 0 not
    int clsize = get2byte(FILE *f, long int file_position, int eoferror)
    {
        int cl;
        if ((cl == EOF) && (eoferror == 1))
        {
            next << "An Error: Premature End of File (in getbyte())!\n";
            exit(1);
        }
        cl = get2byte(FILE *f, long int file_position);
        return cl;
    }
    // Read in an integer from a file. [LSB, MSB]
    long int clsize = get2byte(FILE *f, long int fp)
    {
        int cl;
        cl = get2byte(FILE *f, fp);
        cl = get2byte(FILE *f, fp+1);
        return cl + cl*256;
    }
    // Read in a long integer
    long int clsize = get2byte(FILE *f, long int fp)
    {
        int cl;
        cl = get2byte(FILE *f, fp);
        cl = get2byte(FILE *f, fp+1);
        cl = get2byte(FILE *f, fp+2);
        cl = get2byte(FILE *f, fp+3);
        return cl + cl*256 + cl*65536 + cl*16777216;
    }
}

```

【図150】

```

long int c1, c2;
c1 = get2bytes(f, fp);
c2 = get2bytes(f, fp);
return c1 + c2*65536;
}

// Plot image using Flash Graphics at (xloc, yloc) with color
void cimage::plot(int x, int y)
{
    int px, py;          // Pixel coords on the screen (upper left)
    char color;

    for (py=0; py<sizeR; py++)
        for (px=0; px<sizeC; px++)
        {
            color = pixel[py][px];
            if (color & 1)
                xs_plotxy(x+px, y+py); // Draw it
        }
}

// Plot image with a box around it (with color 1)
void cimage::plotbox(int x, int y)
{
    int px, py;

    for (px=0; px<sizeC+2; px++) { // Draw horizontal lines
        if ((x+px-y)&2 == 0)
            xs_plotxy(x+px, y);
        if ((x+px+y+sizeR-1)&2 == 0)
            xs_plotxy(x+px, y+sizeR-1);
    }
    for (py=1; py<sizeR+1; py++) {
        if ((x+y-py)&2 == 0)
            xs_plotxy(x, y+py);
        if ((x+sizeC-1+y+py)&2 == 0)
            xs_plotxy(x+sizeC-1, y+py);
    }
    plot(x+1, y+1); // Plot the image
}

```

【図167】

```

extern void out_dash(cimage* line, struct Charlist* clist);
out_dash(("line_to_process", clist);
endif

#ifdef COMBINE
charlist_combine(("line_to_process", clist);
endif

#ifdef PRINT_FINAL_CHARLIST
// Print the characterlist
xs_clr_win();
print_charlist(("line_to_process", clist, 10, 10);
xs_flush();
endif

#ifdef DISPLAY
xs_flush();
endif
printf("Displaying Final Segmentation. Enter 0 to continue? ");
int dummy;
dummy = auto_input_int();

Zeroapp_terminate(masterxlist); // *** DELETE ***

```

【図151】

```

// Filename: cimages.h
// This is the header file for cimages.C
//
//
// =====
// Define MEM_DEBUG 1
#include<stream.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

class cimage; // Forward reference

// =====
// CLASS: cimage
// =====
class cimage
{
public:
    int sizeR; // Number of rows (first index of array)
    int sizeC; // Number of cols. (second index of array)
    int allocated_sizeR; // Amount malloced
    int allocated_sizeC; // Amount malloced
    int dpi; // Dots per inch of the image (0 if unknown)
    unsigned char **pixel; // The binary image (2d array)

    cimage(); // Initialize
    ~cimage(); // Destroy
    cimage(int r, int c); // Initialize with a size
    cimage(cimage* temp);

    void reallocate(int R, int C); // Will allocate memory for an image
    void fill(char value, int R, int C); // Fill image with a constant value
    void clip(cimage* image, int R1, int R2, int C1, int C2);
    void clipzero(); // Clips zero space surrounding image
    void readtif(char *filename); // Read in a tiff image
    void plot(int x, int y); // Plot binary image
    void plotbox(int x, int y); // Plot with a box around it

private:
    void allocate(int r, int c); // Allocate memory
    void deallocate(); // Deallocate memory
    int get1byte(FILE *f, long int* file_position, int noerror);
    long int get2bytes(FILE *f, long int* fp);
    long int get4bytes(FILE *f, long int* fp);
};

```


【図153】

```

for (l=c2->bot - 4; l < c2->bot; l++)
    botpos += c1->curve1[l] + c1->curve2[l];

// Compute position and exit
*d13 = botpos - toppos;

if (c1->float) *d13) / ((float) width_small);
if (l < FRAC_VALID_DIST)
    return 1;
else
    return 0;

// This routine is called from comb_dist() with a pair of characters
// than probably isn't an l or f (l/f=1)
// characters. This determines if close enough to be combined.
// The first character
struct Character *c1;
struct Character *c2;
// The second character
int h1;
// Which character is higher
int *d13;
// The last value
int width_small;
// The width of the smaller character
int toppos, botpos;
// Positions of the top and bottoms
int l;
float f;

if (h12 == 1)
    // Case when c1 is higher than c2
    // Find bottom pos of c1
    if ((c1->bot - c1->stop) < 4) // If character very small, use .loc
        botpos = c1->loc;
    else
        botpos = 0;
    for (l=c1->bot - 4; l < c1->bot; l++)
        botpos += c1->curve1[l] + c1->curve2[l];

// Find top pos of c2
if ((c2->bot - c2->stop) < 4) // If character very small, use .loc
    toppos = c2->loc;
else
    toppos = 0;
for (l=c2->stop; l < c2->stop + 4; l++)
    toppos += c1->curve1[l] + c1->curve2[l];

// Compute position and exit
*d13 = botpos - toppos;

f = ((float) *d13) / ((float) width_small);
if (l < FRAC_VALID_DIST)
    return 1;
else
    return 0;

// c1 is higher than c2
// Find top pos of c1
if ((c1->bot - c1->stop) < 4)
    botpos = c1->loc;
else
    botpos = 0;
for (l=c1->stop; l < c1->stop + 4; l++)
    botpos += c1->curve1[l] + c1->curve2[l];

// Find bottom pos of c2
if ((c2->bot - c2->stop) < 4)
    botpos = c2->loc;
else
    botpos = 0;

```

【図154】

```

else
{
    toppos = 0;
    for (i=1; i <= ci->stop - 4; i++)
        toppos += ci->curve[i] * ci->curve[i];
}

// Find bottom pos of c2
if (c2->bot == c2->top)
    botpos = c2->loc; // If character very small, use .loc
else
{
    botpos = 0;
    for (i=c2->bot - 4; i <= c2->bot; i++)
        botpos += ci->curve[i] * ci->curve[i];
}

// Compute position and exit
*d3 = botpos - toppos;
f = ((float) *d3) / ((float) width_small);
if (f < FRAC_VALID_DIST)
    return 1;
else
    return 0;
}

// This routine returns how close two non-height-overlapping characters
// are (h12 must be valid). Computes characters from top of lower one
// to bottom of higher one. Returns 0, bad. 1, good. Also updates d12
int comb_dist (
    struct Character *c1, // The first character
    struct Character *c2, // The second character
    int h12, // This character is higher
    int *d12) // The last value
{
    int width_c1; // The size of c1 (width)
    int width_c2; // The size of c2
    int height_c1; // The height of c1
    int height_c2; // The height of c2
    float f;

    width_c1 = c1->right - c1->left;
    width_c2 = c2->right - c2->left;
    width_small = width_c1;
    width_small = width_c2;

    if (c12 != VERY_SMALL) // If it's been computed, don't recompute it
    {
        f = ((float) *d12) / ((float) width_small);
        if (f < FRAC_VALID_DIST)
            return 1;
        else
            return 0;
    }

    height_c1 = c1->bot - c1->top;
    height_c2 = c2->bot - c2->top;

    if (h12 == 1) // c1 is higher than c2

```

【図155】

```

// This routine will check for the two characters being the 1st
// part of a percent sign o/ Returns 1 for it's a %, 0 for no.
int comb_left_percent(
    struct Character *c1,
    struct Character *c2)
{
    // The characters
    // Check for o/ part of percent (types 2):
    float hl, wl; // The height and width of the line
    float hd, wd; // The height and width of the dot
    int p; // How far from the top to take the slope from
    hl = (float) (c1->bot - c1->stop);
    wl = (float) (c1->right - c1->left);
    hd = (float) (c1->bot - c1->top);
    wd = (float) (c1->right - c1->left);
    if (wl * POW_PERCENT_OVERLAY <= (float) (c1->right - c1->left))
    {
        if (wl * POW_DOT_WIDTH_MIN <= wd)
        {
            // If dot is width is not too small
            if (wl * POW_DOT_WIDTH_MAX >= wd)
            {
                // If dot is width is not too large
                if (hl * POW_DOT_HEIGHT_MIN <= hd)
                {
                    // If dot is height is not too small
                    if (hl * POW_DOT_HEIGHT_MAX >= hd)
                    {
                        // If dot is height is not too large
                        p = (int) (wl * POW_TO_TAKE_SLOPE); // Ignore from top and bottom
                        if ((float) (c1->curved[c1->stop] - c1->curved[c2->bot-p]) /
                            (float) (c1->bot - c1->stop - p - p) >= POW_MIN_SLOPE)
                        {
                            return 1;
                        }
                    }
                }
            }
        }
    }
    return 0;
}

// This routine will check for the two characters being the 2nd
// part of a percent sign o/ Returns 1 for it's a %, 0 for no.
int comb_right_percent(
    struct Character *c1,
    struct Character *c2)
{
    // The characters
    // Check for o/ part of percent (types 2):
    float hl, wl; // The height and width of the line
    float hd, wd; // The height and width of the dot
    int p; // How far from the top to take the slope from
    hl = (float) (c1->bot - c1->stop);
    wl = (float) (c1->right - c1->left);
    hd = (float) (c1->bot - c1->top);
    wd = (float) (c1->right - c1->left);
    if (wl * POW_PERCENT_OVERLAY <= (float) (c1->right - c1->left))
    {
        if (wl * POW_DOT_WIDTH_MIN <= wd)
        {
            // If dot is width is not too small
            if (wl * POW_DOT_WIDTH_MAX >= wd)
            {
                // If dot is width is not too large
                if (hl * POW_DOT_HEIGHT_MIN <= hd)
                {
                    // If dot is height is not too small
                    if (hl * POW_DOT_HEIGHT_MAX >= hd)
                    {
                        // If dot is height is not too large
                        p = (int) (wl * POW_TO_TAKE_SLOPE); // Ignore from top and bottom
                        if ((float) (c1->curved[c2->stop] - c1->curved[c1->bot-p]) /
                            (float) (c2->stop - c1->bot - p - p) >= POW_MIN_SLOPE)
                        {
                            return 1;
                        }
                    }
                }
            }
        }
    }
    return 0;
}

// This routine will slide the list over, reading in a new character
// i.e. 1 <- 2 <- 3 <- 4 <- New, (entire list)
void comb_slide(
    struct CombinaList *comb) // The list
{
    // Slide the characters down
    comb->c1 = comb->c2;
    comb->c2 = comb->c3;
    comb->c3 = comb->c4;
    comb->c4 = comb->next;
    if (comb->next != 0)
    {
        comb->next->comb->next; // Get the next character in the list
        // Slide down heights as well
        comb->h1 = comb->h2;
        comb->h2 = comb->h3;
        comb->h3 = comb->h4;
        // Set to undefined
        comb->h4 = -1;
        // Slice down distances
        comb->d1 = comb->d2;
        comb->d2 = comb->d3;
        comb->d3 = comb->d4;
        comb->d4 = -1;
        // Set to undefined
        comb->d5 = -1;
    }
}

// This routine will combine two chars (which better be adjacent) This
// will be done by moving c1 and adding it on to the end of c1.
void comb_combine(
    struct Character *c1,
    struct Character *c2)
{
    // The first character
    int loc; // The location of the new character
    // The top of the new character
    int ploc; // The location divided by 8 (real position)
    int bot; // The bottom of the new character
    int count; // Used to compute loc
    int al; // Range of overlapping portion of characters
    int left; // The left and right of new character
    int right; // The left and right of new character
    // Combine curved and curved2, and compute top and bot
    if ((c1->bot + c2->stop) < (c2->bot + c1->stop))
    {
        // Nonoverlapping chars
        // Copy curves of c2 into c1
        for (i=c2->stop; i<=c2->bot; i++)
        {
            c1->curved[i] = c2->curved[i];
            c1->curved2[i] = c2->curved2[i];
        }
        // compute new location
        loc = (c1->loc + (c1->stop - c1->bot) * (c2->bot - c2->stop)) /
            (c1->bot - c1->stop + c2->bot - c2->stop);
        ploc = loc/8;
        // Fill blank space between to chars (set to loc, and loc won't change.)
        // (and set top and bot)
        if (c1->bot < c2->stop)
        {
            for (i=c1->bot; i<=c2->stop; i++)
            {
                c1->curved[i] = c1->curved[i];
                c1->curved2[i] = c1->curved2[i];
            }
        }
        top = c1->stop;
    }
}

```

【図156】

```

if (c1->left < c2->left)
    left = c1->left;
else
    left = c2->left;
if (c1->right > c2->right)
    right = c1->right;
else
    right = c2->right;

// Load information about the stuff.
c1->loc = loc;
c1->stop = top;
c1->bot = bot;
c1->next = c1;
c1->right = c2->next;
if (c2->next != 0)
    (c2->next->prev = c1);
c1->type = CHARACTER_CONSIDER;
c1->type = CHARACTER_CONSIDER;
}

// This routine will combine characters 1 and 2, and slide them
void comb_combine(comb->c1, comb->c2)
{
    struct Combinelist *comb; // The list
    comb_combine(comb->c1, comb->c2);
    comb->next = comb;
    comb->slide(comb);
}

// This is the main routine to combine a charlist
void charlist_combine(
    struct Charlist *clist) // Image of the line
{
    struct Combinelist comb; // The 4 characters in the que we're evaluating
    int dummy;

    // Initialize the combination list
    comb->next = c1->first; // Slide in 4 characters into the list
    comb->slide(comb);
    comb->slide(comb);
    comb->slide(comb);
    comb->slide(comb);

    // Main loop: Warning! Indentation is not standard! (but not too bad)
    while(comb->c2 != 0) // loop until list is empty
    {
        // Combine 1 & 2
        if (comb->next->delta(comb->c1, comb->c2, 4(comb->h2)))
            if (comb->next->delta(comb->c1, comb->c2, 4(comb->h2)))
                if (comb->next->delta(comb->c1, comb->c2, 4(comb->h2)))
                    // Combine 1 & 3 instead?
                    if (comb->c3 != 0)
                        if (comb->next->delta(comb->c1, comb->c3, 4(comb->h3)))
                            if (comb->next->delta(comb->c1, comb->c3, 4(comb->h3)))
                                if (comb->next->delta(comb->c1, comb->c3, 4(comb->h3)))
                                    // Compute left and right
    }
}

```

—346—

[illegible]

【图 1 5 9】

[illegible]

【図160】

```

//def PRINT_INT0
int dummy;
endif

histogram = vector<line>(); // Allocate the memory
charon = clist->first; // Loop until end of list
while (charon != 0)
{
    ed_char(line, clist, charon, histogram);
    //def PRINT_INT0
    printf("Enter 0 to continue: ");
    scanf("%d", &dummy);
    if (dummy == 0)
    {
        charon = charon->next; // Get the next character
    }
    free_vector(histogram); // Deallocate the memory
}

// Check for the dash between a character, and break it if needed
void ed_char(
    struct Charlist *clist, // The line
    struct Character *ch, // Pointer to charlist
    int *hist) // To skip past newly inserted divisions
{
    int p;
    // The histogram (previously allocated)

    // Reject if pixel width is too small.
    if ((ch->right - (ch->left)) < MIN_ABS_WIDTH_TO_CONSIDER)
        return;

    //def PRINT_INT0
    x = ch->win();
    print_character(line, "ch, 10, 10);
    x = x + 1;

    printf("\nCharacter info:");
    printf(" top/bot = (%d, %d)\n", (ch->stop, (ch->bot));
    printf(" left/right = (%d, %d)\n", (ch->left, (ch->right));
    printf(" ");
    // Compute the histogram
    //def PRINT_INT0
    // Break left side
    for (p = 0; p < ch->left; p++)
    {
        // Out the character
        //def PRINT_INT0
        printf(" ");
        printf(" top/bot = (%d, %d)\n", (ch->stop, (ch->bot));
        printf(" left/right = (%d, %d)\n", (ch->left, (ch->right));
        // Break right side
        p = ch->right;
        // Out the character
        //def PRINT_INT0
        printf(" ");
    }

    //def PRINT_INT0
    // This will cut the dashes of of characters
    void cut_dashes(line, // The image
    struct Charlist *clist, // The current charlist
    int *histogram, // The area of the histogram
    struct Character *charon, // The character we're processing
    int *p);
}

```

【図161】

```

// filename: level3.c
// This file contains the routines to do level 3 segmentation (slightly
// touching) using method 2 (histogram approach)
//
//
// The includes
//
#include <charlist.h>
#include <charlist.h>
#include <math.h>

// The constants
#define HISTOGRAM_THRESHOLD 10

// This routine will adjust left, right, curvel, top, bottom
// to the character in the character list. It will adjust
// the character list to position the character.
// Possible optimization: Usually, this is called when a character is
// split, so shrinking one side is 'in vels'
void l3l_shrinkup_char(
    struct charlist *ch) // The line
{
    int top, bot, left, right; // Boundaries to the character we're on
    int i; // The new left and right stuff.
    int loc, loc_count, pos; // pos = position, loc = pos - i
    int curvel, curvel_count; // Shortcuts, set to (*ch)-curvel and 2
    int flag;

    curvel = (*ch)-curvel;
    curvel_count = (*ch)-curvel_count;

    // Adjust the top of the character. If it needs to move down
    flag = 0;
    for (top = (*ch)-top; (top < (*ch)-bot) && (flag == 0); top++)
        if (line.pixel[top][c] & 1) // Stop at the first on pixel
            flag = 1;
    break;

    // Adjust the bottom of the character
    flag = 0;
    for (bot = (*ch)-bot-1; (bot > (*ch)-top) && (flag == 0); bot--)
        if (line.pixel[bot][c] & 1) // Stop at the first on pixel
            flag = 1;
    break;

    // Make bot exclusive
    bot++;

    // Now adjust curvel and curvel_count, updating left and right to 'shrink'
    // tightly around the border of the character (move the borders up to the
    // edge of the characters.

```

```

loc = loc_count = 0; // for position of the char (on the fly)
for (i=top; i<bot; i++)
{
    while (curvel[i] < curvel[i+1]) // Loop until they're equal
        if ((line.pixel[i][curvel[i]] & 1) // If there is a non empty space
            // to the right of the left border
            // Then exit (left border not moved)
            break;
    else
        curvel[i] += 1; // There's an empty space to the right
    while (curvel[i] > curvel[i+1]) // Slide the border over to meet it
        if ((line.pixel[i][curvel[i]-1] & 1) // Do the same with the right border:
            break;
    else
        curvel[i] -= 1;
    if (curvel[i] != curvel[i+1]) // there is a pixel on this line...
        loc += curvel[i] - curvel[i+1];
        loc_count++;
}
if (loc_count != 0)
{
    loc = loc * i / loc_count; // Compute the position (used for loc)
    pos = loc / i; // Compute the real position
}

// Find left and right
left = curvel[top];
right = curvel[bot];
for (i=top; i<bot; i++)
{
    if (curvel[i] < left)
        left = curvel[i];
    if (curvel[i] > right)
        right = curvel[i];
}

if (loc == 0)
{
    loc = 4 * (left+right);
    pos = (left+right)/2;
}

for (i=top; i<bot; i++) // Fill in the blank lines with pos
    if (curvel[i] != curvel[i+1])
        curvel[i] = curvel[i+1];

// Copy all the new attributes in
(*ch)-top = top;
(*ch)-bot = bot;
(*ch)-left = left;
(*ch)-right = right;
}

// This routine will cut *ch into two at point p. clist is updated.
// void l3l_cut returns as the *ch points to the right half of the character
// always line
// struct charlist *clist; // The line
// struct character *ch; // The character list
// int p; // Where to cut the character

```

```

// Where the minimum value is
// Left and right boundaries of where to search
int min, min_at;
int l, r;
int c;

// Compute the histogram
histogram[line, ch, hist];

// Start at left side
p_and = ('ch')->right;
// Exit when we've scanned the entire histogram
while (p < p_and)
{
    while (p < p_and)
    {
        // Looking for top-bottom crossing of threshold
        if (hist[p] > histogram_threshold)
        {
            histogram[p] = histogram_threshold;
            break;
        }
        // This is the left side of the area to search
        l = p;
        while (p < p_and)
        {
            // Looking for bottom-top crossing of threshold
            if (hist[p] < histogram_threshold)
            {
                histogram[p] = histogram_threshold;
                break;
            }
            // Remember right edge
            r = p;
            if (p < p_and)
            {
                // If we haven't gone over the edge
                // Search for min in l..r range
                min = hist[l];
                min_at = l;
                for (c=l; c < r; c++)
                {
                    if (hist[c] < min)
                    {
                        min = hist[c];
                        min_at = c;
                    }
                }
                // If the min is small enough, cut there
                {
                    l_min = min_at - histogram_threshold;
                    r_max = min_at + histogram_threshold;
                    // Cut the character
                    // which changes the right char,
                    // which shouldn't mess things up
                    printf("Level 3 made a cut: %d", min_at);
                }
            }
        }
    }
}

// MAIN ROUTINE
// This will do histogram touching on the characters for level 3 method 3
void level3(char* line,
struct Charlist* clist)
{
    // The change
    struct Charlist* clist;
    // The area of the histogram
    int histogram;
    // The character we're processing
    histogram = vector(line, size);
    // Allocate the memory
    change = clist->right;
    while (change != 0)
    {
        // Loop until end of list
        // Insert breaks (line, clist, echrom, histogram, min_cut_threshold,

```

[illegible]

【图 1 6 5】

[illegible]

【图 1 6 6】

[illegible]

```

// File name: multiLevel.C
// This routine contains stuff to do layer 8 segmentation
//
//
// Find Projection at a rotated angle
// Note: This routine is extremely unoptimized for speed and space!
// Input: image, theta,
// Break the breaking, num is the break on the left side to start
// searching from index allocated for
// allocated: max_x = highest_vpp index value (output),
// Output: a_vpp, max_x = highest_vpp index value (output),
// Index on a what point a[0] corresponds to,
// Define P_STORED 10000 // Memory allocated for projection
// Define P_SIZE 10000 // Where the zero is
// void find_projection(image, angle, // The input image,
// double theta, // The angle,
// struct Breaklist* blist, // The breaklist for the image,
// int numx, // break # of left side for computing profile
// int numy, // break # of right side for computing profile
// int c, // right point to search for
// int allocated_s, // size of s allocated
// int max_x, // max x of a[] allocated
// int max_y, // max y of a[] allocated
// int a_max, // Output: What highest index of a used
// int a_max_0, // Output: What point a[0] corresponds to
// int storey_P_STORED, // The profile
// int start_store, stop_store, // The highest values used in store
// int i, //
// int j, //
// int k, //
// int D, //
// int iLeftBreak = new int[image.size()]; // The start and stop locations
// int iRightBreak = new int[image.size()];
// // Initialise for computing the Angular VPP
// for (i=0; i < P_STORED; i++) store[i] = 0; // zero a matrix
// start_store = P_STORED; // Define the bounds to zero
// breaklist_return_curves(blist, numx, numy, leftbreak, rightbreak);
// // Compute the Angular VPP
// for (i=0; i < image.size(); i++) // Loop through image
// for (jLeftBreak[i]; jLeftBreak[i]; j++) // Loop through image
// if (i <= C-(image.size()-jLeftBreak[i])) // If it's outside the line
// break;
// // Edit: from looping through c
// {
// image.pixel[c] += 0;
// }
// p = AVPP_compute_point(c, c, theta); // Calculate point
// if (p < 0) // (p > P_STORED) // Check range
// {
// cout << "Error in find_projection(): P_STORED too small!\n";
// exit(1);
// }
// store[p]++; // Increment count
// if (p < start_store) start_store = p; // Update max and min range
// if (p > stop_store) stop_store = p;
// }
// // Deallocate memory used
// delete (image.size) leftbreak;
// delete (image.size) rightbreak;
// // Copy store to a with information
// if (start_store < P_STORED) // (stop_store > 0) // If no projection
// max_x = 0; // Return size of 0

```

[図169]

```

index_0a = 0; // Set just in case
return; // Exit
}

max_a = stop_store - start_store + 1;
if (max_a > allocated_a)
{
    cerr << "Error in find_projection: "
    << "Not enough memory allocated for matrix A(10^6)";
    exit(1);
}

index_0a = start_store - P_TZERO;
for (i = start_store; i < stop_store; i++) // Copy store to a
    *a(i - start_store) = store[i];

// Convert a double to an int by clipping down:
// i.e. -0.5 -> -1; 1.7 -> 1
int clipdouble(double i)
{
    if (i >= 0)
        return (int) i;
    else
        return (int) i-1;
}

// Computes what column of the rotated histogram point (r,c) falls
// in when the axis is rotated theta degrees
int AVRr_compute_point(int r, int c, double theta)
{
    double dr, dc; // Row and Column
    double di; // Misc. var (length of diagonal to r,c)
    double dp; // Output variable
    double alpha; // Misc. var (angle of (1,0) to (r,c))

    dr = (double) r-.5; // Convert to double
    dc = (double) c-.5; // (the -.5 gets the center of the box)
    alpha = atan(dr/dc);
    di = sqrt(dr*dr + dc*dc);
    dp = di * cos(theta-alpha);
    return clipdouble(dp);
}

// The product routine to do thinning
// Overlap: C = A overlapped with B (times) (must be same dimension)
void product(dimage&C, dimage&A, dimage&B)
{
    int i,j;
    C.fill(0, A.size(), A.size());
    for (i=0; i<A.size(); i++)
        for (j=0; j<A.size(); j++)
            C.pixel(i,j) = A.pixel(i,j) * B.pixel(i,j) / 255.0;
}

// This will do thinning
void thinning(dimage&in_image, double sigma, dimage&out_image)
{
    dimage temp, t1, t2, t3;
    int capsule;

    // Make the gaussian dimage
    capsule = (int) sigma * 1.0;
    temp.size = (int) sigma * 6;
    temp.makegauss(sigma, capsule); // Make dimage

    // Blur degraded image by sigma
    t1.convol(in_image, temp); // t1 = blurred image
    t1.scale(0.255);

    // t2 = derivative of blurred image
    t2.compute_dxt(t1);
    t3.compute_dyt(t1); // t3 = inverted derivative of blurred image
    for (i=0; i<t2.size(); i++)
        for (j=0; j<t3.size(); j++)
            t2.pixel(i,j) = 255.0 - t2.pixel(i,j);

    product(out_image, t2, t3); // t4 = blurred + derivative (product)

    // Plot one line through a point
    // This plots point for point in the y direction (so it is to be
    // void plot_line_candidate()
    int x_point; // The x coord of the point
    double theta; // The angle
    int y1; // The y range to print
    int y2;
    double a, b, xd;
    int x,y;

    // Note: m and b are for x = my + b (11111)
    a = tan(theta); // Compute the slope
    b = -a_point - a*x_point; // Compute intercept

    for (y=y1; y<y2; y++)
    {
        xd = a * (double) y + b;
        x = (int) xd;
        xa_plot(x,y);
    }

    // The Segmentation Routines
    // The breaklist to insert breaks in
    // the VPP (previously calculated)
    // The max value of the VPP
    // What value column proj(0) corresponds to
    void insert_segproj_breaks()
    {
        breaklist.clear();
        int proj;
        int proj_size;
        int proj_start;
    }

```


【図17.1】

```

allocated_s, 4, max_s, index_0a);
// Find maximum projection
max_value_s = 0;
for (i=1; i<max_s; i++)
    max_value_s = max_value_s > x_ploxy(i) ? x_ploxy(i) : max_value_s;

// Choose range to ignore points before first peaks on left and right
r1 = 0;
r2 = max_s-1;

if (theta != 0)
{
    // Find left peak (while slope is still rising), and above LA_IGNORE_HEIGHT
    while ((a1(r1) < a(r1)) || (a1(r1) < a(r2))) || (a1(r1) < a(r2-1)) ||
        (a1(r1) < LA_IGNORE_HEIGHT * max_value_s))
    {
        if ((++r1) == max_s-1) // If got to left edge
        {
            min = min_at = -1; // Return no minimum found
            free_vector(r1);
            return;
        }
    }
    // Find right peak (while slope is falling (from right to left))
    while ((a1(r2) < a(r2)) || (a1(r2) < a(r1))) || (a1(r2) < a(r2-1)) ||
        (a1(r2) < LA_IGNORE_HEIGHT * max_value_s))
    {
        if ((--r2) == r1) // If reached the left point
        {
            min = min_at = -1; // Return no minimum found
            free_vector(r1);
            return;
        }
    }

    find_minimum_in_vector(r1, r2, min, min_at); // Find the minimum

    // ..... The above is mostly copied from void find_min_valley_at_theta ....
    // ..... The following section is VERY VERY poorly written!!!!!!
    for (i=cursor_col_left-1; i=cursor_col_right-1; i++) // Draw line above
        cursor_y += 8;

    // Plot the image
    print_image_with_breaks(image, blist, bnum1, bnum2, cursor_x, cursor_y);

    // Find position of the C line and plot it
    struct break bl; break(blist); // Find initial offset
    int cx = cursor_x - b1.loc/image_size * c;
    int cy = cursor_y + image_size * 2;
    plot_line_at_angle(Q, C, theta, cursor_x - 4, cy + 5); // Right line

    cursor_x += image_size * 5;

    // Plot the histogram
    int ttemp = sample_plotmag_hist(h, max_s, cursor_x, cursor_y);
    sprintf(string, "%g,%g", theta/180.0);
    x_ploxy(cursor_x + max_s * 5, cursor_y + ttemp + 1, string);
    if (theta != 0) // Plot dotted line
    {
        int ttt = (int) (LA_IGNORE_HEIGHT * (double) max_value_s);

```

【図172】

```

// Find p2: the valley to the right of the given peak
px = find_valley_to_right(proj, proj_size, pix, max_valley_depth);
if (px == -1) // If not found, make it the right side
    px = proj[proj_size - 1];
py = proj[proj_size];

// Find p4: the next peak to the right of valley p2
px = find_peak_to_right(proj, proj_size, px, min_peak_height);
if (px == -1) // If not found, make it the right side
    px = proj[proj_size - 1];
py = proj[proj_size];

// Find p3: the nearest valley to the left of peak at p4
px = find_valley_to_left(proj, px, max_valley_depth);
if (px == -1) // If not found, make it the left side
    px = proj[proj_size];
py = proj[proj_size];

thet1 = atan(double) (p2y-p1y) / (double) (p2x-p1x); // This is positive
thet2 = atan(double) (p4y-p3y) / (double) (p4x-p3x); // This is negative

// ===== Mehrtz's optimal threshold finding routine =====
void find_optimal_threshold(
    int proj_size; // The size of proj[]
    int level; // The threshold level
    int* min_valley_depth; // The outputs
    int* min_peak_height; // The outputs
    int max_valley_proj; // The maximum value of the projection
    int i; // Values
    double a, b;
    FILE *fp;
) {
    if ((fp = fopen("constants.dat", "r")) == NULL)
        exit(1);
    cerr << "Warning in opening constants.dat\n";
    fscanf(fp, "%f", &a); // Skip sin char width
    for (i=0; i<level; i++)
        fscanf(fp, "%f", &a, &b); // Read a line
    if ((a <= -1) || (b <= -1)) // Exit
        return; // Return an error
    min_valley_depth = -1;
    min_peak_height = -1;
    return;
}

fclose(fp); // Close the file
// printf("... LEVEL ed = %f, a=%f, level, a, b);
// Find maximum projection (for find peak and valley heights:
// (proj_size <= 0) // Check for size = 0
// max_valley_depth = 0;
// min_peak_height = 0;
return;
}

```

```

// Set to not found
while ((i-pos >= 0) && (peak == -1))
    if ((proj[pos] >= proj[pos+1] && (proj[pos+1] >= min_peak_height))
        peak = pos + 1;
return peak;

// Subroutine to find next valley to the right in a projection
// =====
int find_valley_to_right(
    int proj[], // The projection
    int proj_size, // The max size of the projection (not inclusive)
    int pos, // Where to start looking
    int max_valley_depth) // Highest point a valley can be
{
    int valley; // Where the valley is
    valley = -1; // Set to not found
    while ((i-pos < proj_size) && (valley == -1))
        if ((proj[pos+1] <= proj[pos]) && (proj[pos+1] <= max_valley_depth))
            valley = pos + 1;
    return valley;

// =====
// Subroutine to find next valley to the left in a projection
// =====
int find_valley_to_left(
    int proj[], // The projection
    int pos, // Where to start looking
    int max_valley_depth) // Highest value for the valley
{
    int valley; // Where the peak is
    valley = -1; // Set to not found
    while ((i-pos >= 0) && (valley == -1))
        if ((proj[pos] <= proj[pos+1] && (proj[pos+1] <= max_valley_depth))
            valley = pos + 1;
    return valley;

// =====
// Computes the Mehrtz's angles (given a valley position)
// =====
void find_valley_angles(
    int proj, // The projection vector
    int proj_size, // The size of the projection vector
    int valley_pos, // The position to the right of the 1st peak
    double thet1, // Angle of left wall
    double thet2, // Angle of right wall
    int min_valley_height, //
    int max_valley_depth)
{
    int pix, p1y; // The coord of left peak
    int p1x, p1y; // The coord of left valley bottom
    int p1x, p1y; // The coord of right peak
    int p1x, p1y; // The coord of right valley bottom
    // Find p1: the peak to the left of the valley
    px = find_peak_to_left(proj, valley_pos, min_peak_height);
    if (px == -1) // If none found, make it the left side
        px = 0;
    p1y = proj[p1x]; // Get its height
}

```

【図173】

```

max_value_proj = proj[i];
for (int i = proj.size() - 1; i >= 0; i--)
    if (proj[i] < max_value_proj)
        max_value_proj = proj[i];

// Compute min and max valley heights
min_valley_height = (int) b * (double) max_value_proj;
max_valley_height = (int) b * (double) max_value_proj;

// Send in the maximum character width from the data file
// (get_max_char_width)
int get_max_char_width()
{
    FILE *fp;
    double i;
    if ((fp = fopen("constants.dat", "r")) == NULL)
        error("Error in opening constants.dat");
    fscanf(fp, "%lf", &i); // Read min char width
    fclose(fp);
    return i;
}

// Insert an angular break into the breaklist
void insert_angular_break(
    // The image
    // The breaklist
    // The angular range will be forced to
    // fall within these breaks
    double theta,
    // The break position (for line concatenation)
    int pos)
{
    // The location of the break in the breaklist
    int break_pos;
    // The new break
    int a1, a2;
    // Arrays, breaks for hnum1, and hnum2
    int i;

    break = vector(image.size());
    a1 = vector(image.size());
    a2 = vector(image.size());
    breaklist_return_curves(breaklist, hnum1, hnum2, a1, a2); // Compute break range
    for (i = 0; i < image.size(); i++)
    {
        // Compute break position
        break[i] = (int) ((double) pos / cos(theta) + (double) i * tan(theta));
        // Ensure that it is within range
        if (break[i] < a1(i))
            break[i] = a1(i);
        if (break[i] > a2(i))
            break[i] = a2(i);
    }
}

break_position = (break[0] + break[image.size() - 1]) / 2;
// Insert the break
breaklist_add_break(breaklist, break_position, blist_svector_weight, BREAK_ANGULAR, b);
}

// Free memory
free_vector(breaklist);
free_vector(a1);
free_vector(a2);
}

// Simple routine used by insert_angular_break()
// Checks a value against the current minimum value. If less, it updates.
void insert_angular_compare_min(
    // The current minimum
    int min,
    // The value to test
    double val_theta,
    // It's angle
    int min_theta,
    // The test minimum
    int min_val_theta,
    // etc...
    double min_theta)
{
    if (min_val_theta < 0)
        return;
    if (min_val_theta < min)
    {
        min = min_val_theta;
        min_theta = min_theta;
        min_theta = min_theta;
    }
}

// Don't substitute if min_val_theta == -1
return;
}

// Khebra's angular projection break routine v73
// Inserts angular breaks in a vector image number of breaks inserted
// The image
// The breaklist
// The starting break number
// The stopping break number
// Which set of thresholds to use
int start_break,
int stop_break,
int threshold_level;

int proj_valley_pos,
int proj_valley_depth,
int max_valley_depth,
int max_valley_height,
int i, i2;
double theta1, theta2;
int break_count;

// Used for finding angular projections
// A minimum and where it's located.
// A temporary min and where it's located
int min_valley_pos,
int min_valley_depth,
int min_valley_height,
int min_valley_theta;

// Zero number of breaks
break_count = 0;

// Allocate the projection vector
proj = vector(proj.size());
proj = vector(proj.size());

```

【図174】

```

// Find vertical projection
find_projection(image, 0, blist, start_break, stop_break, image, alisecl,
               proj_alisecl, proj, proj_alise, proj_alisecl, // Compute VPP
               return break_count;

// Find maximum projection (for find peak and valley heights)
max_valley_proj = proj[0];
if (proj[1] > max_valley_proj)
    max_valley_proj = proj[1];

// Compute max_valley_depth & min_peak_height
compute_optimal_threshold(proj, proj_alise, thresh_level,
                           max_valley_depth, min_peak_height);
if (max_valley_depth < 1) // Exit if we're too deep in constraints
    return break_count;

//def VISUAL
// *** PLOT ***
cursor1_x = cursor1_y; // Initialize Cursor2 (going down)
cursor2_x = cursor2_y; // Compute 1 and 2 columns
breaklist = return_break(blist, start_break, stop_break, and);
cursor1_x = [blist - blist] / image_size * 40; // Compute width of column
cursor2_x = [blist - cursor2_y] / image_size * 40; // (and more cursor 3 to right)
for (i = 0; i < image_size; i++) // Plot double vert. column
{
    xa_plotxy(cursor1_x, i, 1);
    xa_plotxy(cursor2_x, i, 1);
}
print_image_with_breaks(image, blist, start_break, stop_break,
                        cursor1_x, cursor2_x, // Print the non-broken image
                        xa_flush());
// *** ENDPLOT ***

// *** PLOT *** // Plot the histogram
int tttt = simple_histogram(proj, proj_alise, cursor1_x, cursor2_y);
int cocol = cursor1_y - tttt - min_peak_height; // Plot thresholds
for (i = 0; i < image_size; i++)
{
    xa_plotxy(i, cursor1_x, cocol);
    xa_plotxy(i, cursor2_x, cocol);
}
int tttt = cursor2_x - cocol;
cursor1_y = tttt - 10;
xa_flush();
// *** ENDPLOT ***

// Find first peak to right
// find_peak_to_right(proj, proj_alise, 0, min_peak_height);
// if peak not found...
return break_count;

// Find each valley, starting from left position
while(1)
{
    // Search for valley to right
    // valley_pos would be the peak on the left
    valley_pos = find_valley_to_right(proj, proj_alise, 1, max_valley_depth);
    if (valley_pos == -1) // if not found
        return break_count;
}

```

```

// Search for next peak to right of valley_pos
// find_peak_to_right(proj, proj_alise, valley_pos, min_peak_height);
// if not found
return break_count;

//def VISUAL
// *** PLOT *** // ENTPLOT
// Now we know a valley exists, so compute angles
find_valley_angles(proj, proj_alise, valley_pos, thetat, thetat2,
                  min_valley_height, max_valley_depth);
// cout << "Position << valley_pos;proj_sero << 'Angles = ' << thetat;150;3;15
150 << ' ' << thetat2;150;3;14159 << '\n';
// Place C at the next peak (on the right)
C = 1;proj_sero;
// Find minimum of the VPP (theta = 0)
find_minimum_in_vector(proj, valley_pos, C, proj_sero, min, min_at);
min_theta = 0;
min_at = proj_sero;
// cout << " VPP minimum at << min_at << ' ' << min << '\n';
// Compute the angular vpp from the last break to C, and find minimum
find_min_valley_at_theta(image, blist, start_break, break_count,
                        min_theta, min_at, thetat, thetat2);
// cout << " At Theta1, min = << min << '\n';
// If it has a smaller minimum, remember it
insert_angular_compare_min(min, min_at, min_theta, min_at, thetat, thetat2);
find_min_valley_at_theta(image, blist, start_break, break_count,
                        min_theta, min_at, thetat, thetat2);
// cout << " At Theta2, min = << min << '\n';
// Insert angular_compare_min(min, min_at, min_theta, min_at, thetat, thetat2);
// Insert_angular_compare_min(min, min_at, min_theta, min_at, thetat, thetat2);
// Check other angles.
double three_degrees = 3.14159/180.0 * 3.0;
double six_degrees = 3.14159/180.0 * 6.0;
double nine_degrees = 3.14159/180.0 * 9.0;
// find_min_valley_at_theta(image, blist, start_break, break_count,
//                        min_theta, min_at, thetat, thetat2);
// cout << " At Theta3, min = << min << '\n';
// Insert_angular_compare_min(min, min_at, min_theta, min_at, thetat, thetat2);
// Check theta1 - 3
// find_min_valley_at_theta(image, blist, start_break, break_count,
//                        min_theta, min_at, thetat, thetat2);
// cout << " At Theta4, min = << min << '\n';
// Insert_angular_compare_min(min, min_at, min_theta, min_at, thetat, thetat2);
// Check theta1 + 4
// find_min_valley_at_theta(image, blist, start_break, break_count,
//                        min_theta, min_at, thetat, thetat2);
// Insert_angular_compare_min(min, min_at, min_theta, min_at, thetat, thetat2);

```

【図175】

```

// Insert angular break (image, blist, start_break_count,
// stop_break_count, min_theta, min_at);
break_count++; // Move 1 to the next peak do another peak
i = i2;
}

// This routine is used as a link to pass a character to recognition
// to see if we need to go to deeper levels or not. It returns a
// 10 to go deeper, 0 otherwise (if know, it's backwards!)
// Parameters: level; // The image
// blist; // The breaklist
// min_theta; // The matrix to send to the recognition routine
// min_at; // The matrix to send to the recognition routine
// Used to calculate size_c
// Return value
// array1, array2;
array1 = vector(image.size);
array2 = vector(image.size);
breaklist_return_curves(blist, bl, bl, array1, array2);
c = 10; // 31000;
for (i=0; i<image.size; i++)
{
    if (array1[i] < 1) // Out range
    {
        array1[i] = 1;
        array2[i] = 0;
    }
    if (array1[i] > 2)
    {
        array1[i] = 2;
        array2[i] = 0;
    }
    if (array1[i] > 2)
    {
        array1[i] = 2;
        array2[i] = 0;
    }
}
size_c = c - 1;
size_x = image.size;
if (size_c > MAX_CHAR_WIDTH) // If too wide, automatic failure
    return 0;
a = matrix(size_x, size_c);
for (i=0; i<size_x; i++)
    for (j=0; j<size_c; j++)
        a[i][j] = 0;
for (i=0; i<image.size; i++)
    for (j=0; j<size_c; j++)
        a[i][j] = 255;
return_value = recognised(a, size_x, size_c);
free(matrix(a, size_x));
return return_value;
}

// At theta=4, min = 0 <= min <= 'un'
// insert angular_compare_min(min, min_at, min_theta, min_at,
// theta=4_degrees);
// Check theta = 4
// find_min_valley_at_theta(image, blist, start_break_count,
// stop_break_count, c, theta=4_degrees, min, min_at);
// cout << " At theta=4, min = 0 <= min <= 'un'";
// insert angular_compare_min(min, min_at, min_theta, min_at,
// theta=4_degrees);
// Check theta = 6
// find_min_valley_at_theta(image, blist, start_break_count,
// stop_break_count, c, theta=6_degrees, min, min_at);
// cout << " At theta=6, min = 0 <= min <= 'un'";
// insert angular_compare_min(min, min_at, min_theta, min_at,
// theta=6_degrees);
// Check theta = 8
// find_min_valley_at_theta(image, blist, start_break_count,
// stop_break_count, c, theta=8_degrees, min, min_at);
// cout << " At theta=8, min = 0 <= min <= 'un'";
// insert angular_compare_min(min, min_at, min_theta, min_at,
// theta=8_degrees);
// Check theta = 10
// find_min_valley_at_theta(image, blist, start_break_count,
// stop_break_count, c, theta=10_degrees, min, min_at);
// cout << " At theta=10, min = 0 <= min <= 'un'";
// insert angular_compare_min(min, min_at, min_theta, min_at,
// theta=10_degrees);
// Check theta = 12
// find_min_valley_at_theta(image, blist, start_break_count,
// stop_break_count, c, theta=12_degrees, min, min_at);
// cout << " At theta=12, min = 0 <= min <= 'un'";
// insert angular_compare_min(min, min_at, min_theta, min_at,
// theta=12_degrees);
// Check theta = 14
// find_min_valley_at_theta(image, blist, start_break_count,
// stop_break_count, c, theta=14_degrees, min, min_at);
// cout << " At theta=14, min = 0 <= min <= 'un'";
// insert angular_compare_min(min, min_at, min_theta, min_at,
// theta=14_degrees);
// Check theta = 16
// find_min_valley_at_theta(image, blist, start_break_count,
// stop_break_count, c, theta=16_degrees, min, min_at);
// cout << " At theta=16, min = 0 <= min <= 'un'";
// insert angular_compare_min(min, min_at, min_theta, min_at,
// theta=16_degrees);
// Check theta = 18
// find_min_valley_at_theta(image, blist, start_break_count,
// stop_break_count, c, theta=18_degrees, min, min_at);
// cout << " At theta=18, min = 0 <= min <= 'un'";
// insert angular_compare_min(min, min_at, min_theta, min_at,
// theta=18_degrees);
// Insert the best break
// cout << " Final at " << min_at << ", theta = " << min_theta << " 180/3.14159 << ",
// value = " << min << "un";
// min_at = min_at;
// find_min_valley_at_theta(image, blist, start_break_count,
// stop_break_count, c, min_theta, min_at, min_at);
// endl;

```

【图 176】

```
// Exit (gone to highest level, and can't cut)
return b;

}

// CONJOINT METHOD 2 FOLLOWS: (The recombining)
if (perform_recombining == 0)
    return b;

// Pass each place to recognition (breaklist range = brum1 to brum1+b)
// Test to see if MAX_CHARS_PER_CHUNK is high enough
if ((brum1-brum1+b-1) >= MAX_CHARS_PER_CHUNK) {
    print("MAX_CHARS_PER_CHUNK in multilevel.c is too small!\n");
    print("( In this case it needs to be at least ", brum1-brum1+b+1);
    exit(1);
}

// Load in all the recognition array, 0 = Not Recognized, 1 = Recognized.
for (l=brum1; l<brum1+b; l++) // Loop through each section
{
    break = Breaklist_return_break_ptr(blist, l);
    if (break==0) // If not previously recognized
        R(l-brum1) = break_sack_recog + pass_to_deeper_level(image, blist, l, 1);
    else
        R(l-brum1) = -1; // End of the Breaklist
}

R(brum1-brum1+b) = -1;

print("Recognize list:\n ");
for (l=0; l<(brum1+b)-brum1; l++)
{
    if (R(l)) == 0)
        printf("%d\n", l);
    else
        printf("%d\n", l);
}

print("\n\n");

print("Goodcut Logic: %s",
      "If R(l)=0 == 0) // If first not recognized ON LEFT EDGE
      ("E N W") // If R(l) == 0)
      ("E N W"),
      "Print('E N W');
      print(" Trying E [N M] = ");
      if (pass_to_deeper_level(image, blist, brum1, brum1+2)) // Try 0-1
          print("Goodcut");
      Breaklist_remove_break(blist, brum1+1); // 0-1 Good
      b += 1;
      break = Breaklist_return_break_ptr(blist, brum1);
      break_sack_recog = 1;
      return b;
      }
      if (R(l) == 1)
          print("bad cut");
          if (R(l) == -1)
              print("E N E Combining E [N M] E and passing lower!");
              break_sack_recog = 1;
              if (pass_0-1_deeper_level(image, blist, brum1, brum1+1), break_sack_recog == -1)
                  b -= 1;
                  b += layer_goodcut(image, blist, brum1, brum1+1, thresh_level, 1);
                  print("Couldn't find a place to insert a break. Exiting goodcut.\n");
}
```


—366—

```

print("Warning: raster segmentation (level 4) has somehow been lost");
print();

// Recognise this piece
if (paste_to_image_level(image, blist, first_break, last_break) != 0)
{
    // It's recognized
    blk = Breaklist_return_break_ptr(blist, 0);
    blk->next_piece = 1;
    free_vector(prof); // Deallocate memory before exiting
    return;
}

// Call the recursive routine, starting at level 0, with recombining
b = layer_get_out(image, blist, first_break, last_break, 0, 1);
free_vector(prof);

void print_image_with_breaks()
{
    // The image
    Breaklist *blist;
    int break1;
    int break2;
    int x;
    int y;

    // Starting value of x
    int start_x = x;
    int start_y = y;
    int start_stop;
    int start_break_b;
    int start_break;
    int wrap_x;

    wrap_x = WINDOW_SIZE_X - 50;

    if (break1 < 0) || (break2 < 0) || (break3 < 0) || (break4 < 0) || (break5 < 0) || (break6 < 0) || (break7 < 0) || (break8 < 0) || (break9 < 0) || (break10 < 0) || (break11 < 0) || (break12 < 0) || (break13 < 0) || (break14 < 0) || (break15 < 0) || (break16 < 0) || (break17 < 0) || (break18 < 0) || (break19 < 0) || (break20 < 0) || (break21 < 0) || (break22 < 0) || (break23 < 0) || (break24 < 0) || (break25 < 0) || (break26 < 0) || (break27 < 0) || (break28 < 0) || (break29 < 0) || (break30 < 0) || (break31 < 0) || (break32 < 0) || (break33 < 0) || (break34 < 0) || (break35 < 0) || (break36 < 0) || (break37 < 0) || (break38 < 0) || (break39 < 0) || (break40 < 0) || (break41 < 0) || (break42 < 0) || (break43 < 0) || (break44 < 0) || (break45 < 0) || (break46 < 0) || (break47 < 0) || (break48 < 0) || (break49 < 0) || (break50 < 0) || (break51 < 0) || (break52 < 0) || (break53 < 0) || (break54 < 0) || (break55 < 0) || (break56 < 0) || (break57 < 0) || (break58 < 0) || (break59 < 0) || (break60 < 0) || (break61 < 0) || (break62 < 0) || (break63 < 0) || (break64 < 0) || (break65 < 0) || (break66 < 0) || (break67 < 0) || (break68 < 0) || (break69 < 0) || (break70 < 0) || (break71 < 0) || (break72 < 0) || (break73 < 0) || (break74 < 0) || (break75 < 0) || (break76 < 0) || (break77 < 0) || (break78 < 0) || (break79 < 0) || (break80 < 0) || (break81 < 0) || (break82 < 0) || (break83 < 0) || (break84 < 0) || (break85 < 0) || (break86 < 0) || (break87 < 0) || (break88 < 0) || (break89 < 0) || (break90 < 0) || (break91 < 0) || (break92 < 0) || (break93 < 0) || (break94 < 0) || (break95 < 0) || (break96 < 0) || (break97 < 0) || (break98 < 0) || (break99 < 0) || (break100 < 0) || (break101 < 0) || (break102 < 0) || (break103 < 0) || (break104 < 0) || (break105 < 0) || (break106 < 0) || (break107 < 0) || (break108 < 0) || (break109 < 0) || (break110 < 0) || (break111 < 0) || (break112 < 0) || (break113 < 0) || (break114 < 0) || (break115 < 0) || (break116 < 0) || (break117 < 0) || (break118 < 0) || (break119 < 0) || (break120 < 0) || (break121 < 0) || (break122 < 0) || (break123 < 0) || (break124 < 0) || (break125 < 0) || (break126 < 0) || (break127 < 0) || (break128 < 0) || (break129 < 0) || (break130 < 0) || (break131 < 0) || (break132 < 0) || (break133 < 0) || (break134 < 0) || (break135 < 0) || (break136 < 0) || (break137 < 0) || (break138 < 0) || (break139 < 0) || (break140 < 0) || (break141 < 0) || (break142 < 0) || (break143 < 0) || (break144 < 0) || (break145 < 0) || (break146 < 0) || (break147 < 0) || (break148 < 0) || (break149 < 0) || (break150 < 0) || (break151 < 0) || (break152 < 0) || (break153 < 0) || (break154 < 0) || (break155 < 0) || (break156 < 0) || (break157 < 0) || (break158 < 0) || (break159 < 0) || (break160 < 0) || (break161 < 0) || (break162 < 0) || (break163 < 0) || (break164 < 0) || (break165 < 0) || (break166 < 0) || (break167 < 0) || (break168 < 0) || (break169 < 0) || (break170 < 0) || (break171 < 0) || (break172 < 0) || (break173 < 0) || (break174 < 0) || (break175 < 0) || (break176 < 0) || (break177 < 0) || (break178 < 0) || (break179 < 0) || (break180 < 0) || (break181 < 0) || (break182 < 0) || (break183 < 0) || (break184 < 0) || (break185 < 0) || (break186 < 0) || (break187 < 0) || (break188 < 0) || (break189 < 0) || (break190 < 0) || (break191 < 0) || (break192 < 0) || (break193 < 0) || (break194 < 0) || (break195 < 0) || (break196 < 0) || (break197 < 0) || (break198 < 0) || (break199 < 0) || (break200 < 0) || (break201 < 0) || (break202 < 0) || (break203 < 0) || (break204 < 0) || (break205 < 0) || (break206 < 0) || (break207 < 0) || (break208 < 0) || (break209 < 0) || (break210 < 0) || (break211 < 0) || (break212 < 0) || (break213 < 0) || (break214 < 0) || (break215 < 0) || (break216 < 0) || (break217 < 0) || (break218 < 0) || (break219 < 0) || (break220 < 0) || (break221 < 0) || (break222 < 0) || (break223 < 0) || (break224 < 0) || (break225 < 0) || (break226 < 0) || (break227 < 0) || (break228 < 0) || (break229 < 0) || (break230 < 0) || (break231 < 0) || (break232 < 0) || (break233 < 0) || (break234 < 0) || (break235 < 0) || (break236 < 0) || (break237 < 0) || (break238 < 0) || (break239 < 0) || (break240 < 0) || (break241 < 0) || (break242 < 0) || (break243 < 0) || (break244 < 0) || (break245 < 0) || (break246 < 0) || (break247 < 0) || (break248 < 0) || (break249 < 0) || (break250 < 0) || (break251 < 0) || (break252 < 0) || (break253 < 0) || (break254 < 0) || (break255 < 0) || (break256 < 0) || (break257 < 0) || (break258 < 0) || (break259 < 0) || (break260 < 0) || (break261 < 0) || (break262 < 0) || (break263 < 0) || (break264 < 0) || (break265 < 0) || (break266 < 0) || (break267 < 0) || (break268 < 0) || (break269 < 0) || (break270 < 0) || (break271 < 0) || (break272 < 0) || (break273 < 0) || (break274 < 0) || (break275 < 0) || (break276 < 0) || (break277 < 0) || (break278 < 0) || (break279 < 0) || (break280 < 0) || (break281 < 0) || (break282 < 0) || (break283 < 0) || (break284 < 0) || (break285 < 0) || (break286 < 0) || (break287 < 0) || (break288 < 0) || (break289 < 0) || (break290 < 0) || (break291 < 0) || (break292 < 0) || (break293 < 0) || (break294 < 0) || (break295 < 0) || (break296 < 0) || (break297 < 0) || (break298 < 0) || (break299 < 0) || (break300 < 0) || (break301 < 0) || (break302 < 0) || (break303 < 0) || (break304 < 0) || (break305 < 0) || (break306 < 0) || (break307 < 0) || (break308 < 0) || (break309 < 0) || (break310 < 0) || (break311 < 0) || (break312 < 0) || (break313 < 0) || (break314 < 0) || (break315 < 0) || (break316 < 0) || (break317 < 0) || (break318 < 0) || (break319 < 0) || (break320 < 0) || (break321 < 0) || (break322 < 0) || (break323 < 0) || (break324 < 0) || (break325 < 0) || (break326 < 0) || (break327 < 0) || (break328 < 0) || (break329 < 
```

【图 180】

[illegible]

—368—

[illegible]

【図182】

```

1: ((fp = fopen("constants.dat", "r")) == NULL)
{
    cerr << "unbError in opening constants.dat\n";
    exit(1);
}

// *** This should change before the final version, when we know the size
// of constants.dat; hardcoded it!
max_thresh_level = 0;
a = vector(200);
b = vector(200);

for (i=0; i<200; i++)
{
    fscanf(fp, "%f", &a[i]);
    if ((i%5) < 0) || (b[i] < 0)
    {
        max_thresh_level = i;
        break;
    }
}

if (i==200)
{
    print("Error: Only 200 entries in constants.dat is allowed! Change the size\n");
    exit(1);
}

Const1 = vector(max_thresh_level);
Const2 = vector(max_thresh_level);
for (i=0; i<max_thresh_level; i++)
{
    Const1[i] = a[i];
    Const2[i] = b[i];
}

free_vector(a);
free_vector(b);

// =====
// Initialize rotation data
// (this must be called before calling compute_rotated_projection)
void i4_initialize_rot()
{
    int d_index; // Index of degree we're working on
    double d_t_d; // The number of degrees, and tangent term
    int r; // Looping vars: row on, and degree on
    int d, b;

    a = NOT_INDEX_RANGE;
    b = MAX_CHAR_HEIGHT;
    rot = matrix(a, b);

    for (d = NOT_DEG_RESOLUTION; d<NOT_MAX_RANGE; d+=NOT_DEG_RESOLUTION)
    {
        d_index = (int) ((d/NOT_DEG_RESOLUTION+.5) - .5);
        d_t_d = tan(d * 3.14159/180.0);
        for (r=0; r<MAX_CHAR_HEIGHT; r++)
        {
            rot[d_index][r] = (int) ((double) r) * b;
        }
    }
}

// ===== PLOTTING ROUTINES =====
void i4_plot_char(
    char *ch, // The image
    int x, // The upper left hand corner (left, top) to plot
    int y)
{
    for (r=ch->stop; r < ch->bot; r++)
    {
        for (c = ch->survel[r]; c<ch->survel[r+1]; c++)
        {
            if (image_pixel[c] & 1)
            {
                x_plotxy(x+ch->left, y+r);
                x_plotxy(x+ch->left+1, y+r);
            }
        }
        x_flush();
    }
}

void i4_plot_charlist(
    charlist *chlist, // The image
    struct Charlist *elist, // The elist
    int x, // The position
    int y)
{
    struct Character *ch; // The x starting position
    int x_start;
    int x_end;
    x_start = x;
    ch = elist->first;
    while(ch != 0)
    {
        if (x + ch->right - ch->left > WINDOW_SIZE_X - 20) // Wrap
        {
            x = x_start;
            y += line_size + 15;
        }
        i4_plot_charlines(ch, x, y);
        x += ch->right - ch->left + 15;
        ch = ch->next; // Get next character
    }
}

// This routine will plot a character in the lower left hand corner
// (for the purposes of displaying to prompt for recognition)
void i4_plot_lower_left(
    char *ch, // The image
    struct Character *ch)
{
    int r;
    int c;
    delta = ch->bot - ch->stop;
    for (c=0; c<ch->right-ch->left; c++)
    {
        for (r=ch->survel[c]; r<ch->survel[c+1]; r++)
        {
            x_plotxy(x+ch->left, y+delta+r);
        }
    }
    for (r=ch->stop; r < ch->bot; r++)
    {
        for (c=ch->survel[r]; c<ch->survel[r+1]; c++)
        {
            x_plotxy(x+ch->left, y+delta+r+2-delta+ch->stop);
        }
    }
    // Plot box around it
}

```

—370—

```

if (n)
    for (i=Cursor_X; i<WINDOW_SIZE_X; i++)
        ms_plotxy(Cursor_X-10,i);
else
    for (i=Cursor_X; i<WINDOW_SIZE_X; i++)
        ms_plotxy(Cursor_X-11,i);
    ms_plotxy(Cursor_X-9,i);
Cursor_X += 40; // Minimum of 40 columns over;
ms_flush();

//***** TEMP RECOGNITION ROUTINE *****
// Recognized character defined by from ch1 to ch2 (inclusive);
// The limit 0 for unrecognized; 1 otherwise;
// Left character
// Right character
struct Character {ch1;
                  // Left character
                  // Right character
};

// Left visible
// Right visible
// Left lower-left-chars (image, ch1, ch2);
send;

// Left PASS TO TOSHI
int size_X, size_Y;
int r,c;
char ch;
size_X = ch1-box - ch1-stop;
size_Y = ch2-right - ch1-stop;
p = matrix(size_X, size_Y);
for (r=0; r<size_X; r++)
    for (c=0; c<size_Y; c++)
        p[r][c] = 0;
for (r=ch1-stop; r<ch2-box; r++)
    for (c=ch1-boxval(r); c<ch2-boxval(r); c++)
        p[r-c-stop][c-ch1-left] = image-pixel(r)[c];
// Left recognized; size_X, size_Y, p)
// Left visible; size_X, size_Y);
}

if ((ch1-right - ch1-stop) > MAX_CHAR_LEN)
    printf("Character unrecognized because it was too long\n");
return 0;
}
printf("Recognised (%s, %s)\n", yes, no);
return auto_input_line();
}

// This routine will determine if a character is too long (and thus exit)
int is_too_long()
{
    struct Character {ch1;
                    // Left character
                    // Right character
};
if ((ch1-right - ch1-stop) > MAX_CHAR_LEN)
    return 1;
return 0;
}

//***** Character Utility Routines *****
// This routine will move ch>left, and ch>right so they are
// set to the 1st and last 'on' pixel in the image. This is used to

```

【図184】

```

int *ang_proj_min, // The min index of ang_proj[] (inclusive) (output)
int *ang_proj_max, // The max index of ang_proj[] (exclusive) (output)
// Note: Projection put in Proj[]

int r; // Looping variables

// Do a standard histogram
if (angle == 0) // Do a standard vpp histogram
{
    ang_proj_min = range;
    ang_proj_max = ch-left;
    ang_proj_max = range;
    if (ang_proj_max > ch-right)
        ang_proj_max = ch-right;
    for (each-left, each-right, c++) // Zero projection array
        ang_proj[c] = 0;
    for (each-stop, each-bot, r++) // Compute the projection
        for (each-curval[r], each-curval[r], c++)
            (ang_proj[c] += 1);
    return;
}

// Do a theta is greater than zero
if (angle > 0) {
    ang_proj_min = range;
    if (ang_proj_min < ch-left)
        ang_proj_min = ch-left;
    ang_proj_max = range;
    if (ang_proj_max > ch-right)
        ang_proj_max = ch-right;
    ang_proj_min = k*(angle)[ch-bot-ch-stop];
    for (each-stop, each-bot, r++) // Zero projection array
        ang_proj[c] = 0;
    for (each-curval[r], each-curval[r], c++) // Compute the projection
        if (image-pixel[c] & 1)
            (ang_proj[c-Mot(angle)[ch-bot-r-1]])++;
    return;
}

// Do a theta is less than zero
angle = -angle - 1;
ang_proj_min = range;
if (ang_proj_min < ch-left)
    ang_proj_min = ch-left;
ang_proj_max = range;
if (ang_proj_max > ch-right)
    ang_proj_max = ch-right;
ang_proj_max = k*(angle)[ch-bot-ch-stop];
for (each-stop, each-bot, r++) // Zero projection array
    ang_proj[c] = 0;
for (each-curval[r], each-curval[r], c++) // Compute the projection
    if (image-pixel[c] & 1)
        (ang_proj[c-Mot(angle)[ch-bot-r-1]])++;
    return;
}

// eliminate white space from the left and right side of a character
// after it is cut so the size won't be too inaccurate.
// Call when place the cut, ch1 = left char, ch2 = right char
void it_cut_image(it)
{
    struct Character *ch1, // The left character (only does right side)
    struct Character *ch2; // The right character (only does left side)

    int r; // The new edges of the characters
    int new_left, new_right;
    // Right side of char1. Look for 1st on pixel
    for (id = ch1-right-1; id < ch1-left; id++)
        for (id = ch1-stop, each-bot, r++)
            if ((id >= ch1-curval[r]) & 1) // If pixel is on
                (new_right = id);
    goto exit_loop;
}

// Break out of the loop
exit_loop:
// Set right side of character to this value
ch1-right = new_right;
for (each-stop, each-bot, r++)
    if (ch1-curval[r] > new_right)
        ch1-curval[r] = new_right;
ch1-curval[id] = new_right;
if (ch1-curval[id] > new_right)
    ch1-curval[id] = new_right;
}

// Do the left side of ch2
// Look for 1st on pixel
for (each-stop, each-right, c++)
    for (each-stop, id < ch2-bot, r++)
        if ((id <= ch2-curval[r]) & 1) // If pixel is on
            (new_left = id);
    goto exit_loop;
}

// Break out of the loop
exit_loop:
// Set the left side of character to this value
ch2-left = new_left;
for (each-stop, each-bot, r++)
    if (ch2-curval[r] < new_left)
        ch2-curval[r] = new_left;
ch2-curval[id] = new_left;
if (ch2-curval[id] < new_left)
    ch2-curval[id] = new_left;
}

//***** THE ROTATED PROJECTION ROUTINES *****
void it_find_projection(
    struct Image, // The input image to find the projection on.
    struct Character *ch, // The character to find the projection on.
    int angle, // The rotation/absolution must be within a
    // The column range to search through
    int range, // The matrix to put the projection into (*MUST*
    int *ang_proj, // be previously allocated)
)

```

【圖 185】

```

// This routine will insert an entry into the PWP cache (A cyclic cache)
// with inputs p1,v1,p2 and outputs loc,angle.
void i_PWP_Cache_Insert( int v1; // The top of last peak
int v2; // The start of valley
int p1; // The top of last peak
int p2; // The end of valley
struct Character *ch; // The final angle we are in
int min_angle; // The final angle
int min_val; // Where the minimum is at
) { if(!def_DISABLE_PWP_CACHE)
return;
endif
int r;
// Make sure the PWP area doesn't have a cut running through it:
for (i=ch-top; i < ch-bot; i++) if (ch-curval[i] < p2))
return; // If so, don't put in PWP_Que
return;
}
if (PWP_Just) // Insert last element
{ PWP_S = 0;
PWP_E = 3;
PWP_Zero = 0;
} else { // List full, swap last out
if ((PWP_S == PWP_E)
// The next line is basically PWP_S-1 + CACHE_SIZE
PWP_S = (PWP_S-1) % PWP_J-PWP_CACHE_SIZE-1; PWP_E-1;
PWP_E = PWP_S;
} else { (PWP_S==0) ? PWP_J-PWP_CACHE_SIZE-1 : PWP_S-1;
}
// Now place the new data at PWP_S:
PWP_D[PWP_S] = v1;
PWP_V1[PWP_S] = v1;
PWP_V2[PWP_S] = v1;
PWP_V3[PWP_S] = v1;
PWP_Z[PWP_S] = p1;
PWP_Loc[PWP_S] = min_val;
PWP_Angle[PWP_S] = min_angle;
}
// This routines will check PWP against past results and if we've done
// it before, it will just return. It won't hit. If hit
int i; // i=PWP_Cache_Hit
// The top of last peak
int v1; // The start of valley
int v2; // The end of valley
int p1; // The top of last peak
int p2; // The end of valley
struct Character *ch; // Output: The final angle
int min_angle; // Output: Where the minimum is at
int min_val;
}
int i,r;
if(!def_DISABLE_PWP_CACHE)
return 0;
endif
// Return if min is
if (PWP_Just)

```


【図186】

```

int proj11,
// The max size of the projection (not inclusive)
int pos,
// Where to start looking
int max_valley_depth // Highest point a valley can be
{
    int valley;
    // Where the valley is
    valley = -1;
    // Set to not found
    while ((pos < proj_size) && (valley == -1))
    {
        if ((proj[pos] <= proj[pos+1]) && (proj[pos+1] <= max_valley_depth))
        {
            return valley;
        }
        pos++;
    }
}

// This routine will find the angles given the p-v-v-p coordinates
// This array is in vvp
void find_valley_angles()
{
    int p1,
    // The top of 1st peak
    int v1,
    // The start of valley
    int p2,
    // The top of 2nd peak
    int v2,
    // The end of valley
    int theta1,
    // Output. The index of angles
    int theta2;
    // (- angles/ROT_DEG_RESOLUTION)

    int p1v1,
    // The height of 1st peak
    int v1v1,
    // The height of left valley bottom
    int v1v2,
    // The height of right valley
    int p2v2,
    // The height of 2nd peak
    double t1, c1,
    // The output angle in radians
    // Get its height
    p1v1 = vvp(p1);
    v1v1 = vvp(v1);
    v1v2 = vvp(v2);
    p2v2 = vvp(p2);

    t1 = atan((double) (v1-p1) / (double) (v1-v1)); // This is positive
    c1 = atan((double) (v1-v1) / (double) (p1-v1)); // This is negative
    t2 = t1 * 180.0 / 3.14159 / ROT_DEG_RESOLUTION;
    c2 = c1 * 180.0 / 3.14159 / ROT_DEG_RESOLUTION;
    theta1 = (int) (t1 * 5);
    theta2 = (int) (c1 * 5);
    // Round the angles
}

// ..... Find minimum in a rotated valley .....
void find_min_valley_at_theta1()
{
    struct Image;
    // The image
    int angle,
    // The angle (rotated)
    int p1,
    // The left range to look through
    int v1,
    // The start of valley
    int min,
    // It's location
    int min_theta;
    // The angle where it's located

    int pmin, pmax;
    // Min and max indices of Proj
    int v1min, v1max;
    // Min and max values of the Proj
    int r1, r2, wvp;
    // Range to search for minimum (inclusive)
    int i, height;
    // The minimum height needed for a peak (so
    // left & right sides won't have a minimum)

    int i;
    // If a zero found!
    if (min == 0)
    {
        return;
        // Don't bother doing the rest!
    }
    // Find the projection (load into Proj())
    find_projection(image, ch, angle, p1, p2, Proj, spin, spinax);
    max_value_proj = Proj(min);
    for (i = min; i < max_value_proj; i++)
    {
        if (Proj[i] > max_value_proj)
        {
            max_value_proj = Proj[i];
        }
    }
    // Choose range to ignore points before first peaks on left and right
    r1 = pmin;
    r2 = pmax;
}

```

【图 187】

```

2) = peak-1);
  // height = (int) ((float) max_value_proj * 12.70836 * HEIGHT * .5);
  // how high to ignore the ends
  // Find left peak (while slope is still rising), and above 12.70836 * HEIGHT
  while ((proj[1] <= proj[2]) || (proj[2] <= 12.70836 * HEIGHT)) {
    if ((i-1) <= peak - 3) // 12 pct to left edge
      return;
    // Find right peak (while slope is falling (from right to left))
    while ((proj[2] <= proj[3-1]) || (proj[3-2] <= proj[3-3]) || (proj[3-2] <= 12.70836 * HEIGHT))
      i++;
    // If reached the left point
    if ((i-2) <= 2) || (i-2 <= -peak-1) // If reached the left point
      return;
    // search this range, and find best minimum
    for (i=i-2; i=i+2; i++)
      if ((proj[i]) <= "min") // Found a better minimum!
        if ((proj[i]) != "min")
          "min = proj[i];"
          "min_peak = i;"
          "min_theta = angle;"
        else {
          if ((fabs(angle) < fabs("min_theta"))
            "min_theta = " + proj[i];
            "min_peak = i;"
            "min_theta = angle;"
          }
          "print('Clearer break')";
        }
    "min_theta = angle;"
  }
}

// ***** The find optimal rotated histogram routines *****
// The insert angular break attempts to insert break in a character at a given
// threshold level. This updates ch-vec and ch-val-vec at for both
// characters in the image.
// Returns number chars (1 or 0)
// The character list to add chars to
// The character to be cut
// The threshold level
// Was on the same character and no break
// was inserted (so we don't have to recompute
// the VPP, etc)

// The start and stop values of the vpp
// The maximum value of the vpp
// Maximum height for a valley
// Maximum height for a peak
// Position of valley
// Position of peak
// Resolution of valley (NOT DTD RESOLUTION)
// Resolution of peak
// The peak-valley peak column numbers
// The location of the minimum
// The location of the maximum

static int vpp_min, vpp_max;
static int max_value_vpp;
int max_valley_depth;
int min_peak_height;
int valley_pos;
int peak_pos;
int valley_theta;
int peak_theta;
int min_theta;
int max_theta;

// The value of the minimum
// Range of angles to search for in theta
// 2nd range of angles (edit!)
// What angle we're finding the projection for
// The new character to be inserted
// (i,j,k,l,s,t,r)
// (replacing_char == 0) // Remember last if reconquering
// 14 find projection (image, ch, 0, ch-left, ch-right, vpp, vpp_min, vpp_max);
// Find maximum projection (for find peak and valley heights)
// max_value_vpp = vpp(vpp_min);
// for (i=0; i<min-1; i++) // min-1 is the start of the vpp
//   max_value_vpp = vpp(i);
// else {
//   print("VPP Remastered: max_value_vpp = " + min + ", max_value_vpp:");
// }
// Compute max_valley_depth and min_peak_height
// max_valley_depth = (int) ((float) min_peak_height - Const(thresh_level));
// min_peak_height = (int) ((float) max_value_vpp - Const(thresh_level));
// if (max_valley_depth <= min_peak_height)
//   min_peak_height++;
// print("max_value_vpp = " + min + ", max_value_vpp:");
// print("thresh_level = " + min + ", thresh_level:");
// print("Const = " + min + ", Const(thresh_level):");
// print("max_valley_depth = " + min + ", max_valley_depth:");
// print("min_peak_height = " + min + ", min_peak_height:");
// ***** END OF PLOTTING *****
// If char visible
// If (repeating_char == 0)
// 1. Plot new column (0);
// 1. Plot character, ch, Cursor_X, Cursor_Y; // Go to a new column
// Cursor_Y = ch-vec[i]; // Plot character
// Cursor_Y = ch-vec[i]; // Plot character
// for (i=vpp_min; i<vpp_max; i++) // Plot vpp
//   for (j=0; j<vpp_max; j++)
//     max_valley(Cursor_X + j, vpp_min, Cursor_Y - j);
// if (Cursor_X < Cursor_Y - vpp_max) // Adjust next col loc
//   max_valley(Cursor_X - Cursor_Y - vpp_max, vpp_min,
//     Cursor_Y - vpp_max);
// Cursor_Y = Cursor_Y + 20;
// End of plotting
// Find first peak to right
// if (find_peak_to_right(vpp, vpp_max, vpp_min, min_peak_height);
//   if (i < 1) // If peak not found...
//     return 0;
// Search for valley to right
// Note: I should be the peak on the left
// if (i < 1) // If valley to right (vpp, vpp_max, 1, max_valley_depth);
//   return 0;
// If not found
// return 0;

```

[illegible]

[189]

```

newchar->rec = UNRECOGNIZED;
ch->rec = UNRECOGNIZED;

14.char_reset_lr(image, ch, newchar); // Update left & right borders
Charlist_insert_after_cut(calist, ch);
return 1;
}

//.....
//..... THE GOSCUR ROUTINES SECTION .....
//.....
//.....
// This routine will cut a character, starting at level 0, and going up
// through the levels until it makes a break. Returns 1 if successful,
// 0 if unsuccessful.
// The level of the break
// The character list to add chars to
// The character to be cut
// The level of the break
// If repeating
// If 1st call is not repeating
// If (insert_angular_break(image, calist, ch, level, repeat))
// return 1;
// We're repeating from now on
// repeat = 1;
// return 0;
// No break inserted.

//.....
//.....
void 14.combine_chars(
struct Charlist *clist,
struct Character *ch1,
struct Character *ch2)
{
struct Character *c;
int i;
if (ch1 == ch2)
return;
if ((ch1->stop != ch2->stop) || (ch1->bot != ch2->bot))
{
printf("Error in 14.combine_chars(): Combining characters who's top's not");
printf("equal. (ch1->stop = %d, ch2->stop = %d), (ch1->bot = %d, ch2->bot = %d)");
exit(1);
}
for (i=ch1->stop; i<ch1->bot; i++)
ch1->newval[i] = ch1->newval[i];
ch1->right = ch1->right;
// Remove all the characters between the two
c = ch2;
while(c != ch1)

```

[illegible]

•

—378—

【図193】

```

/* This program contains common vector and matrix mallocs */
#include<malloc.h>
#include<stdio.h>
/* Routine for error during memory allocation */
void allocation_error(s)
char *s;
{
    int *coredump;
    printf("Memory allocation error %s\nProgram Aborted.\n", s);
    coredump = 0;
    coredump = 0;
}

/*..... IVECTOR .....*/
/* Allocate an integer vector of size [0..n-1] */
int *ivector(n)
int n;
{
    int *v;
    v = (int *)malloc((unsigned) n*sizeof(int));
    if (!v) allocation_error("in ivector()");
    return v;
}

/* Free an integer vector v of size [0..n-1] */
void free_ivector(v)
int *v;
{
    free((int *) v);
}

/*..... FVECTOR .....*/
/* Allocate an float vector of size [0..n-1] */
float *fvector(n)
int n;
{
    float *v;
    v = (float *)malloc((unsigned) n*sizeof(float));
    if (!v) allocation_error("in fvector()");
    return v;
}

/* Free an float vector v of size [0..n-1] */
void free_fvector(v)
float *v;
{
    free((float *) v);
}

/*..... RANGED VECTOR .....*/
/* Allocate a ranged vector v of size [m..n-1] */
int *rvector_ranged(m,n)
int m,n;
{
    int *v;
    v = (int *)malloc((unsigned) (n-m)*sizeof(int));
    if (!v) allocation_error("in rvector_ranged()");
    return (v+m);
}

/* Free an integer ranged vector v of size [m..n-1] */
void free_rvector_ranged(v,m)
int *v;
int m;
{
    free((int *) (v+m));
}

/*..... MATRIX .....*/
/* Allocate an integer matrix of size [0..r-1][0..c-1] */
int **matrix(r, c)
int r,c;
/* The size of the matrix */
{
    int **m;
    int i;
    m = (int **)malloc((unsigned) r*sizeof(int));
    if (!m) allocation_error("in matrix()");
    for (i=0; i<r; i++)
        m[i] = (int *)malloc((unsigned) c*sizeof(int));
    if (!m[i]) allocation_error("in matrix()");
    return m;
}

/* Free an integer matrix m of size [0..r-1][0..c-1] */
void free_matrix(m, r)
int **m;
int r;
/* The size (num rows) */
{
    int i;
    for (i=r-1; i>=0; i--)
        free((int *) m[i]);
    free((int **) m);
}

/*..... CMATRIX .....*/
/* Allocate an char matrix of size [0..r-1][0..c-1] */
char **cmatrix(r, c)
int r,c;
/* The size of the matrix */
{
    char **m;
    int i;
    m = (char **)malloc((unsigned) r*sizeof(char));
    if (!m) allocation_error("in cmatrix()");
    for (i=0; i<r; i++)
        m[i] = (char *)malloc((unsigned) c*sizeof(char));
    if (!m[i]) allocation_error("in cmatrix()");
    return m;
}

/* Free an character matrix m of size [0..r-1][0..c-1] */
void free_cmatrix(m, r)
char **m;
int r;
/* The size (num rows) */
{
    int i;
    for (i=r-1; i>=0; i--)
        free((char *) m[i]);
}

```


【図198】

```

t_turn_right(); // Check area to behind of mouse
if ((t_in_front(line) & 5) == 1) // If something there
{
    t_move_forward(); // Move behind mouse
    continue; // And loop again
}
not_finished = 0; // Else it's a one-pixel image. so exit
line.pixel[t_cursorR][t_cursorC] = line.pixel[t_cursorR][t_cursorC] | 4;
// Make this spot

// Load the info into the Charlist
new_char->stop = top;
new_char->shot = bot; // (Make it exclusive)
new_char->left = left; // (Make it exclusive)
new_char->right = right+1; // (Make it exclusive)
new_char->type = CHARACTER_PURE_OUTLINED;

Charlist_place_new_char(clist); // Insert the new character

// Mark the character as removed from the list
for (r=0; r<line.sizeR; r++)
    for (c=curveL[r]; c < curveR[r]; c++)
        line.pixel[r][c] = line.pixel[r][c] | 4;
}

// =====
// Main routine to call to add breaks to the image
// =====
void do_outlining() // Returns number of characters added
{
    struct Charlist* clist; // The character list
    int start_col; // The range to search through (inclusive)
    int stop_col; // (exclusive)

    int r,c; // The row and column we're on

    for (c=stop_col-1; c>=start_col; c--) // Search through column range
        for (r=line.sizeR-1; r>=0; r--) // Scan from bottom to top
            if (((line.pixel[r][c] & 5) == 1) // On pixel that hasn't been outlined
                {
                    line.pixel[r][c] = 123;
                    print_outlining(line, c);
                    extract_outline_char(line, clist, start_col, stop_col, r, c);
                    // Outline the character
                }
}

```

【図203】

```

/* The routines in this file determine if a character is recognized or not
 * (for segmentation)
 */
#include "x.h" // Include Rick's graphics routines
#include "autoinput.h"

extern int WINDOW_SIZE_Y;

int recognized(a, size_x, size_c) /* Return a 0 for not recognized, 1 for
    recognized */
{
    int **a; // The character matrix a[row][col]
    int size_x; // The number of rows in a
    int size_c; // The number of columns in a

    int r,c; // The row and column of the image
    int dummy; // A dummy variable

    #ifdef VISIBLE
    /* This routine will print the char at the screen's bottom left corner */
    for (r=0; r<size_x; r++) // Loop through the rows
        for (c=0; c<size_c; c++) // Loop through the cols
            if ((r>=0) && (r < size_x) && (c>=0) && (c < size_c))
                if (a[r][c] != 0) // If the pixel is on
                    x_a_plotxy(c+20, r + WINDOW_SIZE_Y - size_x - 20);
            else
                x_a_unplotxy(c+20, r + WINDOW_SIZE_Y - size_x - 20);
            else
                x_a_unplotxy(c+20, r + WINDOW_SIZE_Y - size_x - 20);
    x_a_flush(); // Flush the display buffer (to make sure
    // the character gets printed on the screen)
    #endif

    printf("Character displayed. Enter return value (1 = good, 0 = bad)? ");
    dummy = auto_input_int();
    return dummy;
}

```

—384—

[illegible]

```

int **array;
// Array of histograms: (row)(col)
int **starttext;
// Array of starting text positions
int **stoptext;
// The cursors for the start and stop texts
int **size;
// The size of each list
int **include;
// Include this cursor in the next line (0=no, 1=yes)
int top_div;
// The division with the topmost text
int top_col;
// The text range in the adjacent division (row 1's)
int top_row;
// The text range in the current division (row 1's)
int r1, c1, r2, c2;
// The row and column of the line in page
int subline_height;
// The minimum line height
int subline_width;
// The minimum line width
int i, j, n, b, r, c;
// Nice handy variable

startcol = i-vector(NDIVISIONS);
stopcol = i-vector(NDIVISIONS);
starttext = i-vector(NDIVISIONS);
stoptext = i-vector(NDIVISIONS);
cursor = i-vector(NDIVISIONS);
size = i-vector(NDIVISIONS);
include = i-vector(NDIVISIONS);

min_line_height = (int) ((float) page_dp1 * PDEF_DP1_MIN_BLOCK_SIZE);

////////// Step 1: Define columns and find histograms //////////
// Find range of columns
startcol[0] = 0;
for (i=0; i<NDIVISIONS-1; i++)
{
    c = PAGE_DP1 * (PAGE_DP1 - (i+1)) / NDIVISIONS;
    startcol[i+1] = c - OVERLAP;
    stopcol[i] = c + OVERLAP;
}
stopcol[NDIVISIONS-1] = PAGE_DP1;
// Find histograms
for (i=0; i<NDIVISIONS; i++)
{
    make_partial_histo(page, startcol[i], stopcol[i], ngram[i]);
}

////////// Step 2: Make text area lists //////////
for (i=0; i<NDIVISIONS; i++)
{
    // Make 'zeros' are guaranteed on both ends of the histogram
    size[i] = 0;
    for (j=PAGE_DP1-1; j>=0; j--)
    {
        if (ngram[i][j] == 0) && (ngram[i][j+1] != 0) // if starting text
            size[i]++;
        if (ngram[i][j] != 0 && (ngram[i][j+1] == 0) // if stopping text
            stopcol[i] += size[i];
        size[i]++;
    }
}

////////// Step 3: Find the LLus //////////
for (i=0; i<NDIVISIONS; i++)
{
    cursor[i] = 0; // set all pointers to the top of the list
    end_of_list = 0;
    while(end_of_list == 0)

```

—385—

【図201】

```

////////// Step 3a: Find the highest text
a = 1; page_size;
for (i=0; i<divisions; i++) // Going to be high div
    if (cursor[i] < a) // If valid cursor
    {
        b = startest(i)[cursor[i]] + stopest(i)[cursor[i]];
        if (b < a) // If it's better
        {
            a = b;
            top_div = i;
        }
    }

for (i=0; i<divisions; i++) // Initialize include[]
    include[i] = 1; // Don't include anything
// Include the first one
include[top_div] = 1;

z1 = startest(top_div)[cursor[top_div]]; // Initialize the size of the
z2 = stopest(top_div)[cursor[top_div]]; // change to this segment
z3 = startest(top_div);
z4 = stopest(top_div);

////////// Step 3b: Loop to left and see what columns match
range1 = z1; // Each column to the left
for (i=top_div-1; i>=0; i--) // If we're past the end, don't do it
{
    if (cursor[i] < size(i))
    {
        crange1 = startest(i)[cursor(i)];
        crange2 = stopest(i)[cursor(i)];
        if (overlap(range1, range2, crange1, crange2) > CLOSE_OVERLAP)
        {
            include(i) = 1; // Include this column also
            if (crange1 < z1) // Adjust range of z1, z2, c1, c2
            {
                z1 = crange1;
                z2 = crange2;
            }
            c1 = startest(i);
            c2 = stopest(i);
            range1 = crange1;
            range2 = crange2;
        }
    }
}

////////// Step 3c: Loop to right and see what columns match
range1 = startest(top_div)[cursor(top_div)]; // The adjacent division
range2 = stopest(top_div)[cursor(top_div)]; // Text range.
for (i=top_div+1; i<divisions; i++) // Each column to the left
{
    if (cursor(i) < size(i)) // If we're past the end, don't do it
    {
        crange1 = startest(i)[cursor(i)];
        crange2 = stopest(i)[cursor(i)];
        if (overlap(range1, range2, crange1, crange2) > CLOSE_OVERLAP)
        {
            include(i) = 1; // Include this column also
            if (crange1 < z1) // Adjust range of z1, z2, c1, c2
            {
                z1 = crange1;
                z2 = crange2;
            }
            c1 = startest(i);
            c2 = stopest(i);
            range1 = crange1;
            range2 = crange2;
        }
    }
}

////////// Step 3d: Copy the pieces into the image
// Optimise this: Selectively do fill() and clipset()
line_fill(0, z2-z1, c1-c2); // Fill up the image with zeros
for (i=0; i<divisions; i++)
{
    if (include(i) == 1)
    {
        for (c=startest(i); c<stopest(i); c++)
        {
            for (r=startest(i)[cursor(i)]; r<stopest(i)[cursor(i)]; r++)
            {
                line_pixel(r-z1, c-z1) = (int) page_pixel(r[c],
            }
        }
        line_clipset(i); // Remove white space around the line
    }
}

////////// Step 3e: Do we reject this image?
reject = 0; // Default is accept
if ((z2-z1) < min_line_height)
    reject = 1;
print(" Image range is %d..%d, %d..%d\n", c1, c2, z1, z2);

////////// Step 4f: Process line
if (reject == 0)
{
    clear_image_and_plot_image(line);
    sendf
    print("Enter 0 to run outlining character segmentation, 1 to ignore");
    r = sure_input_int(i);
    if (r == 0)
    {
        process_line(line, clist);
        sendf DEMO_OUTPUT
        sendf seg_plot_line(page, line, clist, c1, c2);
    }
}

////////// Skip over all the used sections
for (i=0; i<divisions; i++) // Up all the cursors
    if (include(i) == 1) // Skip over the included areas
        cursor(i) += 1;

////////// Check if we've gone off the end of the line
end_of_line = 1;
for (i=0; i<divisions; i++) // If the cursor is not at the end...
    if (cursor(i) < size(i)) // ... don't exit
        end_of_line = 0;

free_vector(include);
free_vector(cursor);
free_vector(crange1);
free_vector(crange2);
free_vector(stopest);
free_vector(startest);
free_vector(range1);
free_vector(range2);
free_vector(c1);
free_vector(c2);
}

```

【圖 202】

```

MAX_CHAR_LEN = 42;
print("Enter MAX_CHAR_LEN"); // The maximum character length in pixels
MAX_CHAR_LEN = auto_input_int();

border_ignore = 1;
// cout << "Input border width to ignore (in pixels)";
// cin >> border_ignore;

used_renderer = 0;
auto_init(page);

// Initialize page printing routines

// border_ignore = (int) ((float) page_dpi * DPI_BORDER_IGNORE);
// print("Border ignore = %d\n", border_ignore);

PAGE_B1 = border_ignore; // Define page size
PAGE_B2 = PAGE_B1 - border_ignore;
PAGE_B3 = PAGE_B2 - PAGE_B1;
PAGE_C1 = page_size - border_ignore;
PAGE_C2 = PAGE_C1 - PAGE_C1;

// ..... THIS IS JUST FOR TESTING .....
int i;
x.flush();
// printf("Enter 0 to continue");
i = auto_input_int();

// clear_screen_and_plot_image(page);

printf("Pressing because page is displayed. Input 0 to segment page.");
border_ignore = auto_input_int();

while(1) {
    segment_page(page, x.flush());
    x.flush();
    printf("Program finished. Enter 0 to run again?");
    int ttemp;
    ttemp = auto_input_int();
}

char* terminate_collect;
auto_terminate();

```


【図205】

```

if ((chr->bot - chr->top) > MAX_C_HEIGHT)
    max_c_height = chr->bot - chr->top;
chr = chr->next;
}

//def visible
sepp_bot_row_printed = 1; // Make sure it erases if visible
endif

// Check if we need to replace the entire screen
if ((line_bot - sepp_bot_row_printed) < 0)
    (sepp_cursor = line_bot - 1) > WINDOW_SIZE.Y-10;
    printf("Going to scroll... Enter 0 to continue? ");
    scanf("%d", &i);
    if (i)
        sepp_bot_row_printed = sepp_bot_row_printed - line_bot - 10;
        sepp_cursor = sepp_bot_row_printed - 1; // Get the new cursor
        sepp_cursor = WINDOW_SIZE.Y/2 - 20;
    }

// Print the character list
chr = char->first;
while (chr != 0)
    // Check if need to scroll one line
    if (sepp_cursor < chr->right - chr->left + 10 + WINDOW_SIZE.X)
        sepp_cursor = sepp_cursor - chr->right - chr->left + 10; // Move down
        sepp_cursor = LEFT_EDGE + 50; // Indent
        y = 100; // The Y
        sepp_cursor.y = sepp_cursor; // Initialize cursor
        sepp_cursor = sepp_cursor; // Initialize cursor
        for (i=0; i<MAX_C_HEIGHT; i++)
            for (j=0; j<MAX_C_WIDTH; j++)
                if ((i >= chr->top) && (i <= chr->bot))
                {
                    if ((j <= chr->left) && (j >= chr->right))
                        continue;
                    if ((i <= chr->curval[i]) && (j <= chr->curval[j]))
                        x = plotxy(x, y);
                    x++;
                }
                sepp_cursor = x;
                x = sepp_cursor;
            }
            y++;
        }
        Draw Bar
        sepp_cursor = sepp_cursor;
        for (i=sepp_cursor-2; i<sepp_cursor+2; i++)
            x = plotxy(x, y-2);
            x = plotxy(x, y+2);
        for (i=y-2; i<y+2; i++)
            x = plotxy(sepp_cursor-2, y);
            x = plotxy(sepp_cursor+2, y);

```

【図206】

```

/* This program will test the charlist */
#include "charlist.h"

void print_charlist(cclist)
struct Charlist *cclist;
{
    struct Character *ch;
    printf("Charlist:\n");
    ch = cclist->first;

    while(ch != 0) {
        printf("Loc: %d: Char: (%s) = '%c', ch->loc=%d",
            (ch->prev == 0) ? 0 : ch->prev->top,
            ch->prev == 0 ? "" : ch->prev->top,
            ch->prev == 0 ? "" : ch->prev->top,
            ch->prev == 0 ? "" : ch->prev->top);
        if (ch->next == 0)
            printf("\n");
        else
            printf("%d", ch->next->top);
        printf("\n");
        ch = ch->next;
    }
    printf("End of charlist\n\n");
}

main()
{
    struct Character *ch;
    struct Charlist cclist;
    int a[10]; /* Dummy vector for height */
    int i, p;

    Charlist_initialize(&cclist, 10, 10);

    for (i=0; i<5; i++)
    {
        printf("Position of new Char %d: ", i);
        scanf("%d", &p);
        ch = cclist->chr(cclist->size);
        ch->top = i;
        ch->bot = i+1;
        ch->curve1[i] = ch->curve2[i] = p;
        Charlist_place_new_char(&cclist);
        print_charlist(&cclist);
    }

    while(1)
    {
        printf("Replace which char? ");
        scanf("%d", &i);

        ch = cclist->first;
        while(ch != 0) {
            if (ch->top == i)
                break;
            ch = ch->next;
        }
        if (ch == 0) {
            printf("Character not found.\n");
        } else {
            printf("New position? ");
        }
    }
}

```

【図220】

```

/*
 * Filename: autoinput.h
 * Header file for autoinput.c
 */
.....
/* Function Definitions for mymalloc.c */
#ifdef CPP
/* Definition for C++ files */
extern "C" {
    void auto_initialize();
    int auto_input_int();
    void auto_terminate();
}
#else
/* Definition for C files */
void auto_initialize();
int auto_input_int();
void auto_terminate();
#endif

```

【図223】

```

/*.....*/
void ZeroVpp_terminate(nllist)
struct NonZeroVppList *nllist; /* The list */
{
    free_vector(nllist->list); /* Deallocate memory */
    nllist->allocated = 0; /* Set so won't be accidentally used */
    nllist->size = 0;
}

/*.....*/
void ZeroVpp_add(nllist, a, b)
struct NonZeroVppList *nllist; /* The list */
int a, b; /* Start and stop of the zeroVpp */
{
    if (nllist->size+2 > nllist->allocated) {
        printf("NonZeroVppList FULL!\n");
        exit(1);
    }
    nllist->list[(nllist->size)+1] = a;
    nllist->list[(nllist->size)+2] = b;
}

```

【図207】

```

// Filename: test_mult.C
// This file is used to test and develop multilevel2.C
// =====
#include <stdio.h>
#include "cimage.h"
#include "myalloc.h"
#include "xs.h"
#include "charlist.h"

extern void l4_initialize();
void level4(cimage* line, struct Charlist *clist);

void xs_init()
{
    struct Charlist clist;
    struct Character *ch;
    cimage image;
    int i,j;

    l4_initialize();

    Charlist_initialize(&clist, 10, 300); // 10 chars, 100-max height
    ch = &clist.chr[clist.size]; // Set new_char to the last character

    // =====
    // Load the image
    //
    image.fill(1,10,40);
    image.pixel[6][6] = 0;
    image.pixel[7][6] = 0;
    image.pixel[8][6] = 0;
    image.pixel[6][7] = 0;
    image.pixel[7][7] = 0;
    image.pixel[8][7] = 0;
    image.pixel[6][8] = 0;
    image.pixel[7][8] = 0;
    image.pixel[8][8] = 0;
    image.pixel[6][9] = 0;
    image.pixel[7][9] = 0;
    image.pixel[8][9] = 0;
    image.pixel[2][10] = 0;
    image.pixel[3][10] = 0;
    image.pixel[4][10] = 0;
    image.pixel[5][10] = 0;
    image.pixel[6][10] = 0;
    image.pixel[7][10] = 0;
    image.pixel[8][10] = 0;
    image.pixel[2][11] = 0;
    image.pixel[3][11] = 0;
    image.pixel[4][11] = 0;
    image.pixel[5][11] = 0;
    image.pixel[6][11] = 0;
    image.pixel[7][11] = 0;
    image.pixel[8][11] = 0;
    image.pixel[0][12] = 0;
    image.pixel[1][12] = 0;
    image.pixel[2][12] = 0;
    image.pixel[3][12] = 0;
    image.pixel[0][13] = 0;
    image.pixel[1][13] = 0;
    image.pixel[2][13] = 0;
    image.pixel[3][13] = 0;

    image.pixel[0][14] = 0;
    for (i=1; i<9; i++)
        image.pixel[i][20] = 0;
    for (i=0; i<10; i++)
        image.pixel[i][31] = 0;
    for (i=1; i<9; i++)
        image.pixel[i][22] = 0;
    for (i=2; i<10; i++)
        for (j=2; j<8; j++)
            image.pixel[j][i] = 0;
    for (i=2; i<28; i++)
        for (j=1; j<2; j++)
            image.pixel[j][i] = 0;
    for (i=29; i<31; i++)
        for (j=7; j<9; j++)
            image.pixel[j][i] = 0;

    // Load the character
    ch->top = 0;
    ch->bot = 10;
    ch->left = 0;
    ch->right = 35;
    for (i=ch->top; i<ch->bot; i++) {
        ch->curve1[i] = ch->left;
        ch->curve2[i] = ch->right;
    }

    Charlist_place_new_char(&clist);
    level4(image, &clist);

    printf("**** End of Program ****\n");
}

```

【図235】

```

/*
 * Filename: mymalloc.h
 * Header file for mymalloc.c
 * =====
 */

/* Function Definitions for mymalloc.c */
#ifndef CPP
/* Definition for C++ files */
extern "C" {
    int *ivector(int n);
    void free_ivector(int* v);
    float *fvector(int n);
    void free_fvector(float* v);
    int *ivector_ranged(int n, int m);
    void free_ivector_ranged(int* v, int m);
    int **imatrix(int r, int c);
    void free_imatrix(int** m, int r);
    char **cmatrix(int r, int c);
    void free_cmatrix(char** m, int r);
}
#else
/* Definition for C files */
/* The header file for mallocs */
int *ivector();
void free_ivector();
int *fvector();
void free_fvector();
int *ivector_ranged();
void free_ivector_ranged();
int **imatrix();
void free_imatrix();
char **cmatrix();
void free_cmatrix();
#endif

```

【図240】

```

/* This file prints a character to be recognized on the screen by */
/* calling the routine tishi_recognize() */

char tishi_recognize(size_t, size_t, image)
{
    int size_r; // The number of rows */
    int size_c; // The number of columns */
    char *image; // The image: bit 1 is the image */

    int r,c;
    char input_string[100];

    printf("\nthe character is:\n");
    for (r=0; r<size_r; r++)
    {
        for (c=0; c<size_c; c++)
            if (image[r][c] & 1)
                printf(" "); // On */
            else
                printf("."); // Off */
        printf("\n");
    }
    printf("***** End of character *****\n\n");
    printf("Enter Character? ");
    scanf("%a", input_string);
    return input_string[strlen(input_string)-1];
}

```

【図208】

```

//
// Filename: test_mult.C
// This file is used to test and develop multilevel2.C
//
//
#include <stdio.h>
#include <string.h>
#include "cimage.h"
#include "xmalloc.h"
#include "xs.h"
#include "charlist.h"

extern void l4_initialize();
void level(cimage *img, struct Charlist *clist);

void xs_init()
{
    struct Charlist clist;
    struct Character *ch;
    cimage image;
    int i,j;

    l4_initialize();
    Charlist_initialize(&clist, 500, 300); // 10 chars, 100-max height

    char filename1[100];
    char filename2[100];
    printf("Input filename? /d:/tif/");
    scanf("%s", filename1);
    strcpy(filename2, "/d:/tif/");
    strcat(filename2, filename1);
    image_readtiff(filename2);
    image_clipspace();

    ch = &(clist.charclist.size); // Set new_char to the last character

    // Load the character
    ch->top = 0;
    ch->bot = image.size1;
    ch->left = 0;
    ch->right = image.size2;
    for (i=ch->top; i<ch->bot; i++) {
        ch->curve1[i] = ch->left;
        ch->curve2[i] = ch->right;
    }

    Charlist_place_new_char(&clist);
    level(&image, &clist);

    printf("**** End of Program ****\n");
}

```

【図228】

```

c1 = get2bytes(f, fp);
c2 = get2bytes(f, fp);
return c1 = c2*65536;

// Plot image using Flash Graphics at (xloc, yloc) with color
void cimage::plot(int x, int y)
{
    int px,py; // Pixel coords on the screen (upper left)
    char color;

    for (py=0; py<size1; py++)
        for (px=0; px<size2; px++)
            color = pixel[py][px];
            if (color & 1)
                xs_plotxy(x+px, y+py); // Draw it
}

// Plot image with a box around it (with color 1)
void cimage::plotbox(int x, int y)
{
    int px,py;

    for (px=0; px<size2; px++) { // Draw horizontal lines
        if ((x+px)/142 == 0)
            xs_plotxy(x+px, y);
        if ((x+px+y+size1-1)/142 == 0)
            xs_plotxy(x+px, y+size1-1);
    }
    for (py=1; py<size1-1; py++) {
        if ((x-y+py)/142 == 0)
            xs_plotxy(x, y+py);
        if ((x+size2-1-y+py)/142 == 0)
            xs_plotxy(x+size2-1, y+py);
    }
    plot(x-1, y-1); // Plot the image
}

```

[図210]

```

/* The character to slice bit(s) from */
int p;
/* The beginning of the bit number */
int m;
/* Number of bits to get */
return x >> (p-m); & -(1 << m);
}

/* Routine: ut_ascii_bitmap(width,height,buf)
Description: Display the bitmap using ascii characters
Return: Nothing
Author: Rick Lee
*/
ut_ascii_bitmap(width,height,buf)
int width;
int height;
char *buf;
{
    short bit_code = 0x0000;
    short bit_result = 0;
    short buf_ptr;
    for(x = 0; x < height; x++)
    {
        for(y = 0; y < width; y++)
        {
            bit_code = buf[(x+y)];
            bit_result = bit_code;
            if(bit_result != 0)
                printf("1");
            else
                printf("0");
            bit_code = bit_code >> 1;
        }
        bit_code = 0x0000;
        printf("\n");
    }
}

/* This file contains general utility routines.
COMMENTS:
ut_reverse(width,height,buf)
ut_getbits(x,p,n)
ut_ascii_bitmap(width,height,buf);
Usage:
Just call the routines.
*/
/* ----- Includes ----- */
#include <stdio.h>
#include <stdlib.h>
/* ----- Global ----- */
/* ----- Externs ----- */
/*
Routine: ut_reverse(width,height,buf)
Description: Reverse the bits for display format
Return: Nothing
Author: Rick Lee
*/
ut_reverse(width,height,buf)
int width;
int height;
char *buf;
{
    unsigned char val,new_val;
    int x,y;
    for(x = 0; x < (width-1); x++)
    {
        z = 0;
        val = buf[x];
        new_val = 0;
        for(y = 0; y < height; y++)
        {
            if(ut_getbits(val,z,1))
                new_val = (new_val << 1) | new_val;
            buf[x] = new_val;
        }
    }
}

/* Routine: ut_getbits(x,p,n)
Description: Check if the bit is turned on or not
Return: Nothing
Author: Rick Lee
*/
ut_getbits(x,p,n)

```

【図211】

```

FILE: xs.c
COMMENTS: This file contains routines for displaying 'things'.

xs_flush()
xs_clear(x,y)
xs_unplotxy(x,y)
xs_redisplay()
xs_dis_bitmap(x,y,width,height,bit)
xs_clr_win()
xs_draw_bitmap(x,y,width,height)
xs_redraw()
xs_setcolor(color)
xs_win_dim(win,dch,height)

Usage:
Just call these routines once the hook has been made with the
display routine.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xatom.h>
#include <X11/Xext.h>
#include <X11/Xproto.h>
#include <X11/Xshlib.h>
#include <X11/Xmu.h>

/* ----- Globals ----- */
/* ----- externs ----- */
extern Widget parent; /* The root widget */
extern GC line_gc; /* The display widget */
extern XColor *cmap; /* The graphic context (colors) */

/* ----- ROUTINES ----- */

/* Internal utility to create/return the pixel of an rgb value */
int xs_rgb_color(red, green, blue) /* Returns a colormap entry */
{
    int red, green, blue; /* From 0 to 255 */

    XColor color;
    Colormap cmap;
    Display *dpy = XDisplay(canvas);
    cmap = DefaultColormap(dpy, DefaultScreen(dpy));
    color.red = red*256;
    color.green = green*256;
    color.blue = blue*256;
    if (!XAllocColor(dpy, cmap, &color))
        return (color.pixel);
    printf("Warning: couldn't allocate requested color\n");
}

return BlackPixel(dpy, DefaultScreen(dpy));

/* Routine to set printing color */
void xs_set_foreground(red, green, blue)
{
    int red, green, blue; /* From 0 to 255 */
    XSetForeground(XtDisplay(canvas),
        line_gc, xs_rgb_color(red, green, blue));
}

/* Routine to clear window and set background color */
void xs_set_background(red, green, blue)
{
    int red, green, blue; /* From 0 to 255 */
    XSetBackground(XtDisplay(canvas),
        line_gc, xs_rgb_color(red, green, blue));
    xs_clr_win();
    xs_flush();
}

xs_flush()
{
    /* Flush the queue to X-server */
    XFlush(XtDisplay(canvas));
}

/* Routine: xs_plotxy(x,y)
Description: Draw a point on the screen
Returns: Nothing
Author: Rick Lee
*/
xs_plotxy(x,y)
{
    int x;
    int y;

    XDrawPoint(XtDisplay(canvas), XtWindow(canvas), line_gc, x, y);
}

/* Routine: xs_unplotxy(x,y)
Description: Undraw a point on the screen
Returns: Nothing
Author: Rick Lee
*/
xs_unplotxy(x,y)
{
    int x;
    int y;

    XSetForeground(XtDisplay(canvas), line_gc, xs_rgb_color(0, 0, 0));
    XDrawPoint(XtDisplay(canvas), XtWindow(canvas), line_gc, x, y);
    XSetForeground(XtDisplay(canvas), line_gc, xs_rgb_color(0, 0, 0));
}

```

【図212】

```

/* Routine: xs_redisplay()
Description: Redisplay the "stuff" that was on the window
Return: Nothing
Author: Rick Lee
*/
xs_clr_win()
{
    xcClearArea(XtDisplay(canvas), XtWindow(canvas), 0.0, 0.0, TRUE);
}

/* Routine: xs_draw_box(x, y, width, height)
Description: Draw a box on screen
Return: Nothing
Author: Rick Lee
*/
xs_draw_box(x, y, width, height)
{
    int w;
    int h;
    int width;
    int height;

    XDrawRectangle(XtDisplay(canvas), XtWindow(canvas), line_gc,
        x, y, width, height);
}

/* Routine: xs_string
Description: Draw a string on the screen
Return: Nothing
Author: Rick Lee
*/
xs_string(x, y, str)
{
    int x;
    int y;
    char *str;

    XDrawString(XtDisplay(canvas), XtWindow(canvas), line_gc,
        x, y, str, strlen(str));
}

/* Routine: xs_setcolor
Description: Set the current color
Return: Nothing
Author: Rick Lee
*/
xs_setcolor(color)
{
    int color;

    gc.setForeground(color);
    XSetForeground(XtDisplay(canvas), line_gc, color);
}

/* Routine: xs_clr_win()
Description: Clear the screen
Return: Nothing
Author: Rick Lee
*/
xs_clr_win()
{
    xcClearArea(XtDisplay(canvas), XtWindow(canvas), 0.0, 0.0, TRUE);
}

/* Routine: xs_draw_bitmap(x, y, width, height, buf)
Description: Draw the given bitmap
Return: Nothing
Author: Rick Lee
*/
xs_draw_bitmap(x, y, width, height, buf)
{
    bitmap bitmap;
    pix * The bitmap for display */

    /* Reverse the bit ordering */
    uc_reverse(width, height, buf);

    /* First create bitmap */
    bitmap = XCreateBitmapFromData(XtDisplay(canvas),
        XtWindow(canvas), buf, width, height);

    /* Create the bitmap area */
    pix = XCreateBitmap(XtDisplay(canvas), XtWindow(canvas), width, height, 1);

    /* Now copy the data to bitmap */
    XCopyPixels(XtDisplay(canvas), bitmap, pix, line_gc, 0, 0,
        width, height, 0.0, 1);

    /* Now copy the bitmap to drawing area */
    XCopyArea(XtDisplay(canvas), pix, XtWindow(canvas), line_gc, 0, 0,
        width, height, 0.0, 1);

    /* Now free the bitmap */
    XFreeBitmap(XtDisplay(canvas), bitmap);

    /* Now free the bitmap */
    XFreeBitmap(XtDisplay(canvas), pix);

    /* Now flush the queue to X-server */
    XFlush(XtDisplay(canvas));
}

/* Routine: xs_clr_win()
Description: Clear the screen
Return: Nothing
Author: Rick Lee
*/
xs_clr_win()
{
    xcClearArea(XtDisplay(canvas), XtWindow(canvas), 0.0, 0.0, TRUE);
}

```

【図213】

```

/*
Routine: xs_win_dim
Description: Get the size of the window
Return: Nothing
Author: Rick Lee
xs_win_dim(width,height)
int *width;
int *height;
{
    Arg args1[2];
    Dimension win_width,win_height;

    XtSetArg(args1[0],XtNwidth,&win_width);
    XtSetArg(args1[1],XtNheight,&win_height);
    XtGetValues(canvas,args1,2);

    *width = win_width;
    *height = win_height;
}

/*
xs_act_cmap(w)
Widget w;
{
    int ncolors;
    XColor colors[256];
    Colormap my_cmap;
    int count = 1;

    Display *d;
    int scr;
    Colormap def_colormap;
    XColor color;
    int x;
    int the_color = 3000;

    d = XtDisplay(w);
    scr = DefaultScreen(d);
    ncolors = DisplayCells(d,scr);
    def_colormap = DefaultColormap(d,scr);
    for(x = 0; x < ncolors; x++)
    {
        colors[x].pixel = x;
        colors[x].flags = DoRed | DoGreen | DoBlue;

        colors[x].red = the_color;
        colors[x].green = the_color;
        colors[x].blue = the_color;
        the_color -= 500;
    }
    XQueryColors(d,def_colormap,colors,ncolors);
    my_cmap = XCreateColormap(d,DefaultRootWindow(d),DefaultVisual(d,scr),
    AllocAll);
    XStoreColors(d,my_cmap,colors,ncolors);
}
*/

```

【図245】

```

/* If your routines are in C++ then CPP must be defined in make file */
#ifdef CPP
extern "C" void xs_init();
extern "C" void xs_flush();
extern "C" void xs_plotxy(int x,int y);
extern "C" void xs_unplotxy(int x,int y);
extern "C" void xs_redisplay();
extern "C" void xs_dis_bitmap(int x,int y,int width,int height,char *buf);
extern "C" void xs_clr_win();
extern "C" void xs_draw_box(int x,int y,int width,int height);
extern "C" void xs_string(int x,int y,char *str);
extern "C" void xs_setcolor(int color);
extern "C" void xs_win_dim(int *width,int *height);
#endif

```


【図215】

```

/* File: xc.c
concepts:
This file contains interface to x-display routines.
main()
xc_display(w,data,call_data);
Usage:
Your initialize routine must be xc_init().
#define DEFAULT_WINDOW_SIZE_HEIGHT 400
#define DEFAULT_WINDOW_SIZE_WIDTH 110
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/StringDefs.h>
#include <Xt/Xm.h>
#include <Xt/XmDrawing.h>
#include <Xt/Xt.h>
#include <Xt/Xt.h>
#include <Xt/Xt.h>
#include <Xt/Xt.h>
/* Global Var */
Widget parent;
int start_argc = 1;
GC line_gc;
XColorValues gcw;
/* The GC values */
/* The widget to display */
/* The flag to call init routine */
/* Pointer to GC */
/* The GC values */
/* Externs */
Routines: xc_display(w,data,call_data)
Description: Expose callback
Return: Nothing
Author: Rick Lee
void xc_display(w,data,call_data)
Widget w;
char *data;
XtDrawingAreaCallbackStruct *call_data;
{
    if (start_argc == 0)
    {
        start_argc = 0;
        xc_init();
    }
    main_argc,argv;
    int argc;
    char *argv[1];
    Arg args[10];
}
*/
int n;
int no_set_win_size; /* = 1 to not set window size */
int no_set_win_bg; /* = 1 to not set window background */
if (call the routine to setup C++ interface, take out if no C++ interface */
xc_init();
endif
/* Checks for geometry and background options in command line */
no_set_win_size = 0;
no_set_win_bg = 0;
if (strcmp(argv[1],"-geometry") == 0)
{
    no_set_win_size = 1;
    if (strcmp(argv[2],"-height") == 0)
    {
        no_set_win_size = 1;
        if (strcmp(argv[3],"-width") == 0)
        {
            no_set_win_size = 1;
            if (strcmp(argv[4],"-bg") == 0)
            {
                no_set_win_bg = 1;
            }
        }
    }
}
/* Init the Xt function */
parent = XtInitialize(argv[0],"Xmwp", NULL, 0, argv, n);
/* XGetApplicationResources (parent, 0, NULL, 0, argv, n); */
/* XSetBackground(XtDisplay(parent), line_gc);
/* WhitePixel(XtDisplay(parent));
/* DefaultScreen(XtDisplay(parent));
/* Create drawing area widget */
canvas = XtCreateManagedWidget("canvas",XmDrawingAreaWidgetClass,
parent,NULL,0);
/* Set the GC for display routines */
line_gc = XCreateGC(canvas,0,0,XtDisplay(canvas));
/* Now add callbacks and event handler */
XtAddCallback(canvas,XmExposeCallback,xc_display,NULL);
/* Set the default screen size */
no;
if (no_set_win_size == 0) {
    XSizeArg(args[0],XtWidth,DEFAULT_WINDOW_SIZE_WIDTH);
    XSizeArg(args[1],XtHeight,DEFAULT_WINDOW_SIZE_HEIGHT);
    if (no_set_win_bg == 0)
    {
        XColorValues(gcw);
        XColorValues(canvas,args,0);
    }
    XtDisplay(parent);
/* Display the widget and enter event loop */
XtRealizeWidget(parent);
XtMainLoop();
}

```

【図216】

```

//
// Filename: zerovpp.C
// This routine contains the lines to find blocks that have non-zero vpp
// (level 1 segmentation)
//
// =====
#include "charlist.h"
#include "cimages.h"
// =====
// Scan a line quickly (every 1/4 pixel) looking for an 'on' pixel
int line_scan_quick( // Returns 1 for pixel found, 0 otherwise
    cimage& line, // The line to process
    int c) // The column to scan
{
    int r;
    for (r=0; r<line.sizeR; r++)
        if (line.pixel[r][c] & 1)
            return 1;
    return 0;
}

// =====
// Scan a line completely looking for an 'on' pixel
int line_scan_complete( // Returns 1 for pixel found, 0 for otherwise
    cimage& line, // The line
    int c) // The column to scan
{
    int r;
    for (r=0; r<line.sizeR; r++)
        if (line.pixel[r][c] & 1)
            return 1;
    return 0;
}

// =====
// The main routine. Call this to make the list
void extract_zero_vpp_list(
    cimage& line, // The image of the line to segment
    struct NonZeroVppList *list) // A pointer to the list
{
    int c;
    int left, right;
    for (c=0; c<line.sizeC; c++) // Scan the image
    {
        if (line_scan_quick(line, c) // If found something
        {
            left = right = c; // Lock to the left for the start
            while (--c >= 0)
                if (line_scan_complete(line, c) == 0)
                    break;
            left = c+1;
            c = right; // Keep looking to the right
            while (++c < line.sizeC)
                if (line_scan_complete(line, c) == 0)
                    break;
            right = c;
            zerovpp_add(list, left, right); // Add in the new break
        }
    }
}

```


【図219】

```

/* filename: autoinput.h
 * This file contains the code to input from a filename
 * .....
 */

#include <stdio.h>
#include "autoinput.h"

/* Global variables */
int auto_eof_detected;
FILE *auto_fp;
/* The FILE pointer */
int save_to_file;
/* Saving the input to a file */
int save_count;
/* Number of inputs before newline */
char filename[100];
/* The filename */
/* .....
void auto_initialize()
{
    int not_exit = 1;
    auto_eof_detected = 0;
    while(not_exit)
    {
        printf("Input filename for auto-input? ");
        scanf("%s", filename);
        auto_fp = fopen(filename, "r");
        if (auto_fp == NULL) /* Read open failed */
        {
            auto_fp = fopen(filename, "w");
            if (auto_fp == NULL) /* Write open failed */
            {
                printf("Cannot open file %s\n", filename);
            } else { /* Opened as a new file */
                printf("Enter 1 at the next prompt\n");
                save_to_file = 1;
                auto_eof_detected = 1;
                not_exit = 0;
            }
        } else { /* Read open succeeded */
            printf("Opening existing file %s\n", filename);
            not_exit = 0;
        }
    }

    printf("Do you wish to save the auto-input into this file? (1 = yes) ? ");
    save_to_file = auto_input_int();
    printf("Hehheh! Change the 1st '1' to a '0' to turn off appending to the file\n");
    printf("Hehheh! Change the 1st '1' to a '0' to turn off appending to the file\n");
    save_count = 0;
}

/* .....
void auto_input_int()
{
    int i;
    if (auto_eof_detected)
    {
        printf("\n[Manual input (-9 = newline, -9999 = exit)] ? ");

```

【図221】

```

/* Filename: charlist.c
 * This file contains Charlist and Enterovplist utility routines
 * .....
 */

/* .....
 * By Christopher Sherrick
 * Includes: charlist.h
 * Includes: malloc.h
 * .....
 */

/* .....
 * Call this routine once only
 * void Charlist_initialize(charlist, clist_max, vector_height)
 * struct Charlist *clist; /* The Charlist
 * int clist_max; /* The max number of characters
 * int vector_height; /* Height of the lines (to allocate curves)
 * int i; /* A looping var
 * .....
 */

/* Allocate memory for characters */
clist_max = (struct Character *) malloc((unsigned) clist_max * sizeof(struct Character));
if (clist_max == 0)
    printf("Memory Allocation Error in Charlist_initialize()\n");
}

clist_size = 0; /* Number of characters
clist_allocated = clist_max; /* Current allocated
clist_size = 0; /* No first character
clist_vector_height = vector_height;

/* Allocate memory for curves and set pointers */
/* NOTE: No pointers, hardcode the array into the code */
for (i = 0; i < clist_max; i++)
    clist[i].curve = (vector *vector_height);
    clist[i].curve = (vector *vector_height);

/* .....
 * Zero a previously allocated Charlist
 * void Charlist_zero(charlist)
 * struct Charlist *clist;
 * .....
 */

clist_size = 0; /* Number of characters
clist_first = 0; /* No first character

/* .....
 * Terminate the Charlist
 * void Charlist_terminate(charlist)
 * struct Charlist *clist;
 * int i;
 * for (i = 0; i < clist_max; i++) { /* Free curves
 * free_vector(clist[i].curve);
 * free_vector(clist[i].curve);
 */
}

/* .....
 * Insert a character after doing a cut
 */

```

[図222]

```

/* This routine will insert the character beyond the charlist (chr[size]).
 * loc is recalculated for both character and the new character, however
 * loc is NOT used to place the character. It will
 * place the new character after character. After finished, it is an
 * extremely good idea to reset the breakline.
 */
void Charlist_insert_after(charlist, ch, after)
{
    struct Charlist *charlist; /* The character */
    struct Character *ch_after; /* Pointer to the new character adding */
    struct Character *newchar; /* Set newchar */

    newchar = (struct Character *) malloc(sizeof(struct Character));
    /* Calculate the position of the new character */
    Charlist_recalculate_loc(newchar);
    Charlist_recalculate_loc(charlist);
    /* Check for full list */
    if (charlist->size == 0) /* Increment the size of the list */
    {
        charlist->size++;
        if (charlist->size >= charlist->allocated) /* If too big... */
        {
            printf("Error in Charlist_place_new_char(): Character list is FULL!\n");
            exit(1);
        }
        newchar->prev = ch_after;
        newchar->next = charlist->next;
        charlist->next = newchar;
        if (newchar->next != 0)
            newchar->next->prev = newchar;
    }

    /* This routine will remove a character from the character list
     * - locs. The memory is NOT reallocated!!! So don't rely on this too much.
     * - Also, prev and next pointers of ch are NOT altered!
     */
    void Charlist_remove(Charlist: ch)
    {
        struct Charlist *charlist;
        struct Character *ch;

        /* Remove the character from the list */
        if (ch->prev == 0) /* If it's the first item on list */
            charlist->first = charlist->next;
        else
            charlist->first->next = ch->next;
        if (ch->next != 0)
            ch->next->prev = ch->prev;
        ch->next->prev = ch->prev;
    }

    /* This routine will calculate the position of a character */
    void Charlist_recalculate_loc(struct Charlist *charlist, struct Character *ch)
    {
        struct Character *ch; /* The character */

        int i, loc;
        /* Calculate the position of the new character */
        loc = 0;
        for (i = 0; i < charlist->size; i++) /* Position: average position... */
            loc += ch->curval[i] * ch->curval[i]; /* ... of the curve */
        loc = loc * 4 / (ch->next - ch->top);
        ch->loc = loc;
    }

```

```

/* This routine will place a character (chr) in the charlist.
 * - The routine will place a character (chr) in the charlist.
 * - This routine will place a character (chr) in the charlist.
 * - This routine will place a character (chr) in the charlist.
 */
void Charlist_place_character(Charlist: charlist, ch)
{
    struct Charlist *charlist; /* The character */
    struct Character *ch; /* The character */
    struct Character *p; /* Used for searching for location */
    struct Character *last_p; /* The character before p */

    /* Is the list empty? */
    if (charlist->size == 0)
    {
        ch->next = 0;
        ch->prev = 0;
        charlist->first = ch;
        return;
    }

    /* Possible optimization: don't start from start of list...
     * start from where it */
    loc = ch->loc; /* Retrieve position */

    /* Is it at the beginning of the list? */
    if (loc == charlist->first)
    {
        /* Set the list on the list to this */
        charlist->first = ch;
        return;
    }

    /* Set the pointers */
    p = charlist->first;
    while (p->loc <= loc)
    {
        /* Remember last position */
        last_p = p;
        p = p->next;
        if (p == 0)
            break;
    }

    /* Insert new position after last_p */
    ch->prev = last_p;
    ch->next = last_p->next;
    last_p->next = ch;
    if (p == 0)
        p->prev = ch;
}

/* Zero Vop lists stuff
 * void Charlist_initialize_lists(struct Charlist *charlist)
 * {
 *     struct Character *p; /* The list */
 *     int size; /* The number of entries (2x allocated) */
 *
 *     /* Allocate memory for list */
 *     size = charlist->size; /* Compute memory allocated, etc */
 *     size = size * 2; /* Double it */
 *     charlist->memory = (struct Character *) malloc(size * sizeof(struct Character));
 *     charlist->allocated = size; /* Set to zero elements */
 *     charlist->size = 0;
 * }

```

—403—

[illegible]

【図225】

```

// pixel = unsigned char **malloc((unsigned int) size*sizeof(unsigned char)); //
// Allocate memory
if (pixel == NULL) // Allocate memory
{
    cerr << "Error: Memory Allocation Error (out of memory)\n";
    exit(1);
}
for (int i = 0; i < size, i++)
{
    pixel[i] = (unsigned char *)malloc((unsigned int) size * sizeof(unsigned char));
    // Allocate row mem
    if (pixel[i] == NULL)
    {
        cerr << "Error: Memory Allocation Error (out of memory)\n";
        exit(1);
    }
    allocated_size = size;
    allocated_sizec = sizec;
}

// Deallocate memory for an image from memory
void cimage::dealloc()
{
    if ((allocated_size != 0) && (allocated_sizec != 0))
    {
        for (int i = 0; i < allocated_size; i++)
        {
            free(pixel[i]);
            if (i % 100 == 0) // Zero out the sizes
            {
                size = 0;
                sizec = 0;
                allocated_size = 0;
                allocated_sizec = 0;
            }
        }
    }

    // Deallocate memory from outside the class. It will
    // Allocate memory if it's larger. If not, it will just adjust the size
    void cimage::realloc(int R, int C)
    {
        int new_R, new_C;
        if ((R > allocated_size) || (C > allocated_sizec))
        {
            new_R = allocated_size;
            new_C = allocated_sizec;
            if (R > new_R)
            {
                new_R = R;
            }
            if (C > new_C)
            {
                new_C = C;
            }
            allocated_size = new_R;
            allocated_sizec = new_C;
        }
    }

    // Make a cimage with a single value: value = value, X and Y are the size
    void cimage::fill(char value, int R, int C)
    {
        // This file contains a visc array image utility functions, for a cimage
        // //define PRINT_IMAGES_INFO
        // //undefine this to make images.c run invisibly (except for error)
        // #include "visc.h"
        // ... CLASS: cimage (in array of chars)
        // .....
        cimage::cimage()
        {
            size = 0;
            sizec = 0;
            allocated_size = 0;
            allocated_sizec = 0;
            dpi = 0;
            allocate(r,c);
        }

        cimage::cimage(int r, int c)
        {
            size = 0;
            sizec = 0;
            allocated_size = 0;
            allocated_sizec = 0;
            dpi = 0;
            allocate(r,c);
        }

        // so problem won't be caused when
        // // allocate called dealloc
        cimage::cimage(int r, int c)
        {
            size = 0;
            sizec = 0;
            allocated_size = 0;
            allocated_sizec = 0;
            dpi = 0;
            allocate(r,c);
        }

        // Allocate memory for new cimage (loads size)
        for (int i = 0; i < size; i++)
        {
            pixel[i] = new char[sizec];
        }

        // On the cimage
        cimage::cimage()
        {
            dealloc();
        }

        // Allocate memory for an cimage from memory
        void cimage::allocate(int r, int c)
        {
            if (r < 0 || c < 0)
            {
                cerr << "Illegal cimage size: " << r << " x " << c << "\n";
                exit(1);
            }
            size = r;
            sizec = c;
            if ((r != 0) && (c != 0))
            {
                // Remove current cimage
                dealloc();
            }
        }
    }
}

```


【図226】

```

allocate(R, C); // Make the new image (note size adjusted)
for (int i=0; i<sizeR; i++)
    for (int j=0; j<sizeC; j++)
        pixel[i][j] = value;
}

// Copy a section of a image into another. image = reference image
// Range is R1..R2 (C1..C2)
void change_clip_image(image, int R1, int R2, int C1, int C2)
{
    int r, c, t;

    if (R2 < R1) // Swap R1 & R2
    {
        t = R1; R1 = R2; R2 = t;
    }
    if (C2 < C1) // Swap C1 & C2
    {
        t = C1; C1 = C2; C2 = t;
    }
    if (R2 < 0) R2 = 0; // Check limits
    if (R1 < 0) R1 = 0;
    if (C2 < 0) C2 = 0;
    if (C1 < 0) C1 = 0;
    if (R1 > image->sizeR) R1 = image->sizeR;
    if (R2 > image->sizeR) R2 = image->sizeR;
    if (C1 > image->sizeC) C1 = image->sizeC;
    if (C2 > image->sizeC) C2 = image->sizeC;

    allocate(R2-R1, C2-C1);
    dol = image->dol;
    for (r=R1; r<R2; r++)
        for (c=C1; c<C2; c++)
            pixel[r][c] = image->pixel[r-R1][c-C1];
}

// ClipSpec: reduce the size of an image by removing the blank
// space around it. This adjusts the image size.
void change_clip_spec()
{
    int R1, R2, C1, C2; // The scanning and stopping rows of new image
    int i, j;
    struct change temp; // Temporary image
    // Initialize to extremes
    R1 = 0;
    R2 = 0;
    C1 = 0;
    C2 = 0;

    // Compute dimensions of new image
    for (i=0; i<sizeR; i++)
        for (j=0; j<sizeC; j++)
            if (pixel[i][j] != 0)
            {
                if (i < R1) R1 = i; // Left edge?
                if (i > R2) R2 = i; // Right edge?
                if (j < C1) C1 = j; // Top edge?
                if (j > C2) C2 = j; // Bottom edge?
            }
    // Check for no pixel found
    if (R1 > R2) R1 = R2 = 0;
    if (C1 > C2) C1 = C2 = 0;

    temp.allocate(R2-R1, C2-C1); // New image is this size
    for (i=R1; i<R2; i++) // Copy clipped image

```

```

        for (j=C1; j<C2; j++)
            temp.pixel[i-R1][j-C1] = pixel[i][j];
        allocate(temp.sizeR, temp.sizeC); // Copy temp image to current image
        for (i=0; i<sizeR; i++)
            for (j=0; j<sizeC; j++)
                pixel[i][j] = temp.pixel[i][j];
    }

    // Read in part of a TIFF file
    void change_read_tiff(char *filename)
    {
        FILE *tagfile; // Input file
        long int tagfile_position; // Tagfile position (in bytes)
        long int width; // Width of picture
        long int length; // Length of picture
        long int black; // Value of black pixel
        long int bps; // Bytes per strip
        long int header_offset; // Start of image data
        long int strips, tag_type, len, value; // Tag values
        long int i, j;

        if (!tagfile = fopen(filename, "rb")) == NULL
        {
            cout << "Error in opening " << filename << "\n";
            exit(1);
        }
        tagfile_position = 0;

        // Read in header and make sure it's Intel
        i = getbytes(tagfile, tagfile_position);
        if (i != 0x002A)
        {
            cout << "Error: TIFF file not in Intel format\n";
            exit(1);
        }

        // Read in Magic number
        i = getbytes(tagfile, tagfile_position);
        if (i != 0x002A)
        {
            cout << "BAD Magic Number for TIFF file!\n";
            exit(1);
        }

        // Find first file directory
        i = getbytes(tagfile, tagfile_position);
        while(tagfile_position < j)
        {
            i = getbytes(tagfile, tagfile_position);
        }

        // Tags in Directory
        numtags = getbytes(tagfile, tagfile_position);
        while (--numtags >= 0)
        {
            tag = getbytes(tagfile, tagfile_position);
            type = getbytes(tagfile, tagfile_position);
            len = getbytes(tagfile, tagfile_position);
            value = getbytes(tagfile, tagfile_position);
            if ((len % 4) && (tag & 32768) == 0)
            {
                cout << "Error: tag length != 1/4\n";
                exit(1);
            }
        }
    }

```

【図227】

```

switch (tag)
{
    case 0x007f: // NewSubFileType
    {
        if (value != 0)
            cout << "Can't deal with Non-Standard NewSubFileType!\n";
        exit(1);
    }
    case 0x007f: // NewSubFileType
    {
        if (value != 1)
            break;
        cout << "Subfile Type not full resolution!\n";
        exit(1);
    }
    case 0x0100: // Width
    {
        width = value;
        break;
    }
    case 0x0101: // Length
    {
        length = value;
        break;
    }
    case 0x0102: // Bits per sample
    {
        if (value != 1)
            cout << "Error: Bits/Sample must be 1!\n";
        exit(1);
    }
    case 0x0103: // Compression
    {
        if (value != 1)
            cout << "Compression Used! Error!\n";
        exit(1);
    }
    case 0x0106: // Black Is
    {
        black = value;
        break;
    }
    case 0x0107: // Thresholding tag
    {
        if (value < 1 || (value > 3))
            cout << "Thresholding Tag = illegal value!\n";
        exit(1);
    }
    case 0x0111: // Offset to Raster Data
    {
        raster_offset = value;
        break;
    }
    case 0x0117: // Bits = value;
    {
        bits = value;
        break;
    }
    case 0x011a: // Bits = value;
    {
        bits = value;
        break;
    }
    case 0x011b: // Bits = value;
    {
        bits = value;
        break;
    }
    case 0x011c: // Bits = value;
    {
        bits = value;
        break;
    }
    default:
    {
        if (value < 0x0100 || value > 0x011c)
            cout << "Warning: An unsupported tag appears in the TIF file!\n";
        break;
    }
} // switch
} // while
}

```

```

while(tagfile_position + raster_offset)
{
    i = getbytes(tagfile, tagfile_position);
    //----- READ IN FILE
    allocate(int) length, (int) width;
    int v;
    for (i = 0; i < length; i++)
        for (j = 0; j < width; j++)
        {
            if (i % 4096 == 0)
            {
                v = getc(tagfile);
                tagfile_position++;
                if (v == EOF)
                {
                    cout << "An Error: Premature End of File found in clif file!\n";
                    exit(1);
                }
                if (v < 128 || v > 255)
                {
                    pixel[i][j] = 0;
                }
                else
                {
                    pixel[i][j] = 1;
                    v = v < 128 ? 1 : 255 - v;
                }
            }
        }
    fclose(tagfile);
}
// Read in a byte. File position is updated
// so far as i to error on EOF, 0 not
int cl_image::getbytes(FILE *f, long int file_position, int sofar)
{
    int ci;
    if (fgetc(f))
    {
        if (fgetc(f) != EOF)
        {
            cout << "An Error: Premature End of File in getbytes()\n";
            exit(1);
        }
        return ci;
    }
    // Read in an integer from file (LSB, MSB)
    long int cl_image::getbytes(FILE *f, long int fp)
    {
        int ci, c2;
        ci = getc(f);
        c2 = getc(f);
        return ci + c2 * 256;
    }
    // Read in a long integer
    long int cl_image::getbytes(FILE *f, long int fp)
    {
        long int ci, c2;
    }
}

```

【図229】

```

// Filename: cimages.h
// This is the header file for cimages.C
//
// =====
#define XEM_DEBUG
#include <array.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

class cimage; // Forward Reference
// =====
/** CLASS: cimage
// =====
class cimage
{
public:
    int sizeR; // Number of rows (first index of array)
    int sizeC; // Number of cols. (second index of array)
    int allocated_sizeR; // Amount allocated
    int allocated_sizeC; // Amount allocated
    int dpi; // Dots per inch of the image (0 if unknown)
    unsigned char *pixel; // The binary image (2d array)

    cimage(); // Initialize
    ~cimage(); // Destroy
    cimage(int R, int C); // Initialize with a size
    cimage(cimage& temp);

    void reallocate(int R, int C); // Will allocate memory for an image
    void fill(char value, int R, int C); // Fill image with a constant value
    void clip(cimage& image, int R1, int R2, int C1, int C2);
    void clipmask(c); // Clips zero space surrounding image
    void readtiff(char *filename); // Read in a tiff image
    void plot(int x, int y); // Plot binary image
    void plotbox(int x, int y); // Plot with a box around it

private:
    void allocate(int R, int C); // Allocate memory
    void deallocate(); // Deallocate memory
    int getbyte(FILE *f, long int& file_position, int eoferror);
    long int get2bytes(FILE *f, long int& fp);
    long int get4bytes(FILE *f, long int& fp);
};

```


【図232】

```

// Copy image into matrix p and call toshl_recognize
if (row_bot == -1) { (row_top == -1)}
{
    size_f = row_bot - row_top;
    size_c = col_right - col_left;
    p = matrix(size_f, size_c);
    for (c=0; c<size_c; c++)
        pixel[c] = (*line_to_process)[c-col_left];
    what_char((f)) = toshl_recognize(size_f, size_c, p);
    free_matrixID, size_f;
}
}
endif

// Ifdef VISIBLE
// Ifdef PASS_TO_TOSH
cursor_y += 10; print_mvpplist_char((*line_to_process), cclist,
    "char_cursor_A.cursor_f");
false
cursor_y += 10; print_mvpplist((*line_to_process), cclist, cursor_f, cursor_y);
x_flush();
endif
Printf("Displaying final segmentation. Enter 0 to continue: ");
int dummy;
dummy = scanf_int();
Zstrcpy_terminate(masterlist); // ... DELETE ...
Zstrcpy_terminate(courierlist); // ... DELETE ...
}

```

```

// No logic (INSERT NOTHING HERE IF NEEDED)
line_to_process = *input_line; // The default
struct Monitorvpplist masterlist; // ... DELETE ...
Zstrcpy_initialize(masterlist, 100);
mlist = masterlist; // ... ADD THIS HIGHEN ...
struct Monitorvpplist courierlist; // ... DELETE ...
Zstrcpy_initialize(courierlist, 100);
cclist = courierlist; // ... ADD THIS HIGHEN ...
endif VISIBLE
x_flush(); // The screen coordinates
cursor_y = 10;
(*line_to_process).plot(cursor_f, cursor_y);
x_flush();
cursor_f += (*line_to_process).sizec * 10;
endif

// Insert breaks caused by zero values of VPP
mlist->size = 0; // Zero the list
extract_zero_vpp_list((*line_to_process), mlist);
// Insert breaks from the courier cut routine
courier_cut((*line_to_process), mlist, cclist);

// Ifdef PASS_TO_TOSH
int size_f, size_c;
int row_bot, row_top;
char **p;
int row_top, row_bot;
int col_left, col_right;
for (i=0; i<cclist->size; i++)
{
    col_left = cclist->list[i];
    col_right = cclist->list[i+1];
    // Find top and bot rows (skip over blank lines)
    row_top = row_bot = -1;
    for (r=0; r<(*line_to_process).sizec; r++)
    {
        if (col_left <= col_right; r++)
        {
            row_top = r;
            row_bot = r;
        }
        for (f=(*line_to_process).sizec-1; f >= 0; f--)
        {
            if (col_left <= col_right; f--)
            {
                row_bot = f;
            }
        }
    }
}

```

【図233】

```

/* This program contains common vector and matrix mallocs */
#include <alloc.h>
#include <stdio.h>
/* Routine for error during memory allocation */
void allocation_error(
    char *s) /* The error text */
{
    int *coredump;
    printf("Memory allocation error %s\nProgram Aborted.\n", s);
    coredump = 0;
    coredump = 0;
}

/*..... VECTOR .....*/
/* Allocate an integer vector of size [0..n-1] */
int *vector(int n) /* Size of vector */
{
    int *v;
    v = (int *)malloc((unsigned) n*sizeof(int));
    if (!v) allocation_error("in vector()");
    return v;
}

/* Free an integer vector v of size [0..n-1] */
void free_vector(v)
    int *v; /* The vector */
{
    free((int *) v);
}

/*..... FVECTOR .....*/
/* Allocate an float vector of size [0..n-1] */
float *fvector(int n) /* Size of vector */
{
    float *v;
    v = (float *)malloc((unsigned) n*sizeof(float));
    if (!v) allocation_error("in fvector()");
    return v;
}

/* Free an float vector v of size [0..n-1] */
void free_fvector(v)
    float *v; /* The vector */
{
    free((float *) v);
}

/*..... RANGED VECTOR .....*/
/* Allocate a ranged vector v of size [m..n-1] */
int *vector_ranged(int m, n) /* The range */
{
    int *v;
    v = (int *)malloc((unsigned) (n-m)*sizeof(int));
    if (!v) allocation_error("in vector_ranged()");
    return (v+m);
}

/* Free an integer vector v of size [m..n-1] */
void free_vector_ranged(v, m, n) /* The range */
{
    free((int *) (v-m));
}

/*..... MATRIX .....*/
/* Allocate an integer matrix of size [0..r-1][0..c-1] */
int **matrix(int r, c) /* The size of the matrix */
{
    int **m;
    int i;
    m = (int **)malloc((unsigned) r*sizeof(int *));
    if (!m) allocation_error("in matrix()");
    for (i = 0; i < r; i++)
        m[i] = (int *)malloc((unsigned) c*sizeof(int));
    if (!m[i]) allocation_error("in matrix()");
    return m;
}

/* Free an integer matrix m of size [0..r-1][0..c-1] */
void free_matrix(m, r) /* The matrix */
{
    int **m; /* The size (num rows) */
    int i;
    for (i = r-1; i >= 0; i--)
        free((int *) m[i]);
    free((int **) m);
}

/*..... CMATRIX .....*/
/* Allocate an char matrix of size [0..r-1][0..c-1] */
char **cmatrix(int r, c) /* The size of the matrix */
{
    char **m;
    int i;
    m = (char **)malloc((unsigned) r*sizeof(char *));
    if (!m) allocation_error("in cmatrix()");
    for (i = 0; i < r; i++)
        m[i] = (char *)malloc((unsigned) c*sizeof(char));
    if (!m[i]) allocation_error("in cmatrix()");
    return m;
}

/* Free an character matrix m of size [0..r-1][0..c-1] */
void free_cmatrix(m, r) /* The matrix */
{
    char **m; /* The size (num rows) */
    int i;
    for (i = r-1; i >= 0; i--)
        free((char *) m[i]);
}

```

[illegible]

-413-

[illegible]

【図238】

```

line fill 0, r2-cl, c2-cl); // Fill up the image with zeros
for (i=0; i<NOVISIONS; i++)
{
    if (include[i] == 1)
    {
        for (j=startcol[i]; j<stopcol[i]; j++)
            for (k=startrow[i]; k<stoprow[i]; k++)
                line_pixel(r2-cl, c2-cl) = (int) page_pixel(r1-cl, c1-cl);
        line_clip_space(); // Remove white space around the line
    }
}

// Step 3a: Do we reject this image?
reject = 0; // Default is accept
if (r1-cl < MIN_line_h(pbc))
    reject = 1;
printf("image range is %d..%d, %d..%d, %d..%d, %d..%d, %d..%d, %d..%d\n", r1-cl, c1-cl, r2-cl, c2-cl, r1-cl, c1-cl, r2-cl, c2-cl);

// Step 4a: Process line
if (reject == 0)
{
    if (reject == 0)
    {
        clear_scale_and_plot_image(line);
        printf("Enter 0 to run character segmentation, 1 to ignore? ");
        if (auto_input_int(i))
        {
            if (i == 0)
            {
                process_line(line);
                step_plot_line_page, line, clist, r1, r2);
            }
        }
    }
}

// Step 4b: Loop to left and see what column match
range1 = r1;
range2 = r2;
for (i=0; i<NOVISIONS; i++)
{
    if (include[i] == 1)
    {
        for (j=startcol[i]; j<stopcol[i]; j++)
            for (k=startrow[i]; k<stoprow[i]; k++)
                line_pixel(r2-cl, c2-cl) = (int) page_pixel(r1-cl, c1-cl);
        line_clip_space(); // Remove white space around the line
    }
}

// Step 4c: Loop to right and see what column match
range1 = r1;
range2 = r2;
for (i=0; i<NOVISIONS; i++)
{
    if (include[i] == 1)
    {
        for (j=startcol[i]; j<stopcol[i]; j++)
            for (k=startrow[i]; k<stoprow[i]; k++)
                line_pixel(r2-cl, c2-cl) = (int) page_pixel(r1-cl, c1-cl);
        line_clip_space(); // Remove white space around the line
    }
}

// Step 4d: Copy the pieces into the image
// Optimize this: selectively do fill() and clip_space()

```

【図239】

```

int border_ignore;
int i;

Charlist_initialize(&clist, MAX_CHARS_PER_LINE, MAX_LINE_HEIGHT);
Process_line_fill(0,10,1500); // Allocate the memory for the lines

xs_win_dim(&WINDOW_SIZE_X, &WINDOW_SIZE_Y); // Get the window size

printf("Input tiff filename? /d2/tif/");
strcpy(filename, "/d2/tif/toshii.tif");
scanf("%s", filename);
strcpy(filename, "/d2/tif/");
strcat(filename, filename);

auto_initialize(); // Initialize Auto-Input routines

printf("Enter courcut average width? ");
COURCUT_AVG_WIDTH = auto_input_int();

printf("Enter courcut delta? ");
COURCUT_DELTA = auto_input_int();

page.cpi = 300;

border_ignore = 1;
// cout << "Input border width to ignore (in pixels)? ";
// cin >> border_ignore;

printf("Reading in File... ");
page.readtiff(filename);
printf("Done.\n");

// segp_init(page); // Initialize page printing routines
// border_ignore = (int) ((float) page.cpi * DPI_BORDER_IGNORE);
// printf("Border ignore = %d\n", border_ignore);

PAGE_R1 = border_ignore; // Define page size
PAGE_R2 = page.sizeR - border_ignore;
PAGE_R3 = PAGE_R2 - PAGE_R1;
PAGE_C1 = border_ignore;
PAGE_C2 = page.sizeC - border_ignore;
PAGE_C3 = PAGE_C2 - PAGE_C1;

// ----- THIS IS JUST FOR TESTING -----
xs_flush();
printf("Enter 0 to continue? ");
i = auto_input_int();

// clear_scale_and_plot_image(page);

printf("Pausing because page is displayed. Input 0 to segment page? ");
border_ignore = auto_input_int();

while(1) {
    segment_page(page, &clist);
    xs_flush();

    printf("Program finished. Enter 0 to run again? ");
    ist ttemp;
    ttemp = auto_input_int();
}

Charlist_terminate(&clist);
auto_terminate();

```

—416—

```

/* FILE: util.c
COMPILER:
This file contains general utility routines.
ut_reverse(width,height,buf);
ut_getbit(x,y);
ut_ascii_stringp1ch,height,buf);
Usage:
    Just call the routines.
*/
#include <stdio.h>
#include <stdlib.h>
/* -----Global-----
/* -----Externs-----
/* Routines: ut_reverse(width,height,buf)
Description: Reverse the bits for display format
Return: Nothing
Author: Rick Lee
ut_reverse(width,height,buf)
int width;
int height;
char *buf;
{
    unsigned char val,new_val;
    int x,y;
    for(x = 0; x < (width*3)*height; x++)
    {
        y = 0;
        val = buf[x];
        new_val = 0;
        for(y = 0; y < 8; y++)
        {
            if (ut_getbit(val,x+y))
                new_val = 0x01 << y | new_val;
        }
        buf[x] = new_val;
    }
}
/* Routines: ut_getbit(x,y)
Description: Check if the bit is turned on or not
Return: Nothing
Author: Rick Lee
ut_getbit(x,y)

```

—417—

[illegible]

【図243】

```

XCopyArea(XtDisplay(canvas), pix, XtWindow(canvas), line_gc, 0, 0,
width, height, x, y);

/* Now free the bitmap */
XFreePixmap(XtDisplay(canvas), bitmap);

/* Now free the bitmap */
XFreePixmap(XtDisplay(canvas), pix);

/* Now flush the queue to X-server */
XFlush(XtDisplay(canvas));

/*
Routine: xs_clr_win()
Description: Clear the screen
Return: Nothing
Author: Rick Lee
*/
xs_clr_win()
{
    XClearArea(XtDisplay(canvas), XtWindow(canvas), 0, 0, 0, 0, TRUE);
}

/*
Routine: xs_draw_box(x, y, width, height)
Description: Draw a box on screen
Return: Nothing
Author: Rick Lee
*/
xs_draw_box(x, y, width, height)
int x;
int y;
int width;
int height;
{
    XDrawRectangle(XtDisplay(canvas), XtWindow(canvas), line_gc,
x, y, width, height);
}

/*
Routine: xs_string
Description: Draw a string on the screen
Return: Nothing
Author: Rick Lee
*/
xs_string(x, y, str)
int x;
int y;
char *str;
{
    XDrawString(XtDisplay(canvas), XtWindow(canvas), line_gc,
x, y, str, strlen(str));
}

/*
Routine: xs_setcolor
Description: Set the current color
Return: Nothing
Author: Rick Lee
*/
xs_setcolor(color)
int color;
{
    gc = XGCValues(XtDisplay(canvas), line_gc, color);
}

/*
Routine: xs_win_dim
Description: Get the size of the window
Return: Nothing
Author: Rick Lee
*/
xs_win_dim(width, height)
int *width;
int *height;
{
    Arg args[2];
    Dimension win_width, win_height;
    XtSetArg(args[0], XtWidth, win_width);
    XtSetArg(args[1], XtHeight, win_height);
    XtGetValues(canvas, args, 2);
    *width = win_width;
    *height = win_height;
}

/*
xs_set_cmap(w)
Widget w;
{
    int ncolors;
    XColor colors[256];
    Colormap my_cmap;
    int count = 1;

    Display *d;
    int scr;
    Colormap def_colormap;
    XColor Color;
    int x;
    int the_color = 2000;

    d = XtDisplay(w);
    scr = DefaultScreen(d);
    ncolors = DisplayCells(d, scr);
    def_colormap = DefaultColormap(d, scr);
    for(x = 0; x < ncolors; x++)
    {
        colors[x].pixel = w;
    }
}

```

【図246】

```

/* FILE: xt.c
COMMENTS:
This file contains interface to t-display routines.
main():
xt_display(v,data,call_data);
Usage:
Your initialise routine must be xt_init().
*/
-----Includes-----
#include <stdio.h>
#include <stdlib.h>
#include <xt/intermal.h>
#include <xt/xt.h>
#include <xt/xt.h>
#include <xt/xt.h>
#include <xt/xt.h>
#include <xt/xt.h>
#include <xt/xt.h>
#include <xt/xt.h>
#include <xt/xt.h>
/* Global variables -----
Widget parent; /* The root widget */
Canvas; /* The widget to display */
int start_up = 1; /* The flag to call init routine */
cc_line_gv; /* Pointer to cc */
forvalues gv; /* The cc values */
/* -----Externs-----
Routines: xt_display(v,data,call_data);
Description: expose callback
Return: Nothing
Author: Rick Lee
void xt_display(v,data,call_data);
char *data; /* The widget associated with display */
xtdrawingAreaCallbackStruct *call_data; /* Not used */
{
    if (start_up)
    {
        start_up = 0;
        xt_init();
    }
    main(argv,argv);
    char *argv[1];
    {
        Arg args[10];
        int n=0;
        int background;
        int foreground;
        /* Arguments to k-server */
        /* Number of arguments */
        /* Default background and foreground color */
    }
}
*/
/* Call the routine to setup C++ interface, take out if no C++ interface */
xt_init();
/* Init the Xt function */
parent = XtInitialize(argv[0], "P1omap", NULL, 0, &argv);
/* Create drawing area widget */
Canvas = XtCreateManagedWidget("canvas", xtdrawingAreaWidgetClass,
parent, argv[1], n);
/* Set the background and foreground color */
XtSetArg(args[0], XtBackground, foreground);
XtSetArg(args[1], XtBackground, background);
XtSetValues(Canvas, args, 2);
/* Now add callbacks and event handler */
xtdrawingAreaCallbackStruct *callback = XtNew(callbackStruct);
/* Display the widget and enter event loop */
XtRealizeWidget(parent);
/* Set the cc for display routines */
gv.foreground = foreground;
gv.background = background;
/* Set the data */
line_gv = XtNew(xtdrawingAreaCallbackStruct);
defaultRootWindow(XtDisplay(Canvas));
GCForground | GCBackground | GC;
XtMainLoop();
}

```

【図247】

```

//
// Filename: zerovpp.c
// This routine contains the lines to find blocks that have non-zero vpp
// (level 1 segmentation)
//
// =====
#include "charlist.h"
#include "cinages.h"
// =====
// Scan a line quickly (every 1/4 pixel) looking for an 'on' pixel
int line_scan_quick( // Returns 1 for pixel found, 0 otherwise
cinages line, // The line to process
int c) // The column to scan
{
int r;
for (r=0; r<line.sizeR; r++)
if (line.pixel[r][c] & 1)
return 1;
return 0;
}

// =====
// Scan a line completely looking for an 'on' pixel
int line_scan_complete( // Returns 1 for pixel found, 0 for otherwise
cinages line, // The line
int c) // The column to scan
{
int r;
for (r=0; r<line.sizeR; r++)
if (line.pixel[r][c] & 1)
return 1;
return 0;
}

// =====
// The main routine. Call this to make the list
void extract_zero_vpp_list(
cinages line, // The image of the line to segment
struct NonZeroVppList *list) // A pointer to the list
{
int c;
int left, right;
for (c=0; c<line.sizeC; c++) // Scan the image
{
if (line_scan_quick(line, c)) // If found something
{
left = right = c; // Look to the left for the start
while (--c >= 0)
if (line_scan_complete(line, c) == 0)
break;
left = c+1;
c = right; // Keep looking to the right
while (++c < line.sizeC)
if (line_scan_complete(line, c) == 0)
break;
right = c;
zerovpp_Add(list, left, right); // Add in the new break
}
}
}

```

フロントページの続き

(72)発明者 シン・ヤン ワング
 アメリカ合衆国 カリフォルニア州
 92626, コスタメサ ブルマン ストリー
 ト 3188 キヤノン インフォメーション
 システムズ インク. 内

(72)発明者 メザードゥ アール. バエズイー
 アメリカ合衆国 カリフォルニア州
 92626, コスタメサ ブルマン ストリー
 ト 3188 キヤノン インフォメーション
 システムズ インク. 内

(72)発明者 クリストファー エー. シェリック
 アメリカ合衆国 カリフォルニア州
 92626, コスタメサ ブルマン ストリー
 ト 3188 キヤノン インフォメーション
 システムズ インク. 内